

## robota

Oliwier Bogdański, Kacper Szponar

Github: <https://github.com/GacusPL/SWIM-Projekt>

Film prezentujący działanie robota: <https://youtu.be/suzh82IGbKQ?si=6QheTJSNaSfraqPB>

### 1. Opis robota

Robot to dwukołowy autonomiczny pojazd mobilny z jednym dodatkowym kołem obrotowym dla stabilizacji, oparty na mikrokontrolerze STM32F303VCT6 Discovery. Napęd realizowany jest przez dwa silniki elektryczne z przekładniami sterowane za pomocą sygnałów PWM oraz sterownika silników L293D. Skręt wykonywany jest różnicowo – poprzez zmianę prędkości obrotowej poszczególnych silników. Układ zasilany jest autonomicznie przez trzy ogniw Li-Ion typu 18650 każde o napięciu 3,6 V. Do zasilania mikrokontrolera, który pracuje z maksymalnym napięciem 5 V, zastosowano przetwornicę napięcia LM2596HVS, umożliwiającą obniżenie napięcia do bezpiecznego poziomu. Robot wyposażony jest w moduł Bluetooth HC-05, który umożliwia sterowanie robotem za pomocą aplikacji. Sterowanie robotem za pomocą Bluetooth jest realizowane za pomocą aplikacji Arduino Bluetooth control (android). Do wykrywania linii zastosowano moduł z ośmioma czujnikami odbiciowymi KTIR0711S, które pozwalają robotowi poruszać się wzdłuż wyznaczonej trasy np. czarnej linii. Robot zawiera dwie dodatkowe funkcjonalności. Wykrywanie przeszkód oraz odtwarzanie muzyki. Do wykrywania przeszkód na drodze wykorzystano czujnik ultradźwiękowy HC-SR04, który mierzy odległość od obiektów na podstawie czasu powrotu fali ultradźwiękowej odbitej od przeszkody. Druga funkcjonalność to odtwarzanie muzyki, realizowana jest ona za pomocą mini odtwarzacza MP3 z gniazdem microSD (DFPlayer), wzmacniacza audio PAM8403 oraz dwóch głośników 3W. Szacowany rozmiar robota to około 200 x 140 x 65 mm.

### 2. Elementy wybrane do budowy robota

**Podwozie:** Chassis Rectangle 2WD 2-kołowe podwozie robota, oraz obrotowe koło podporowe



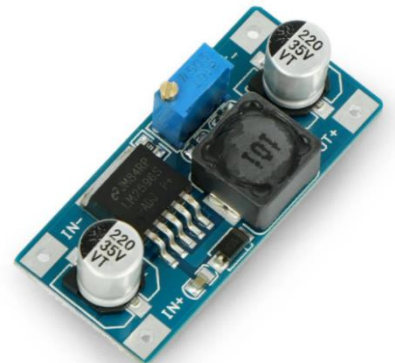
robota

Oliwier Bogdański, Kacper Szponar

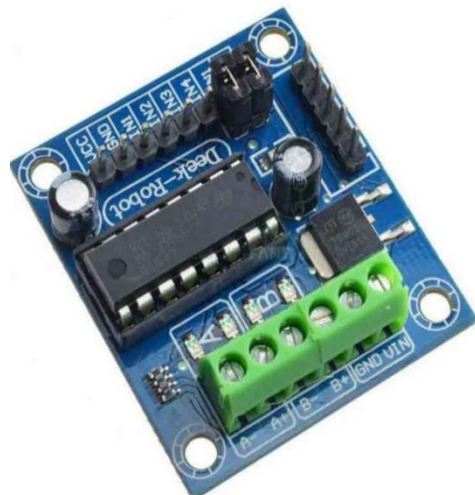
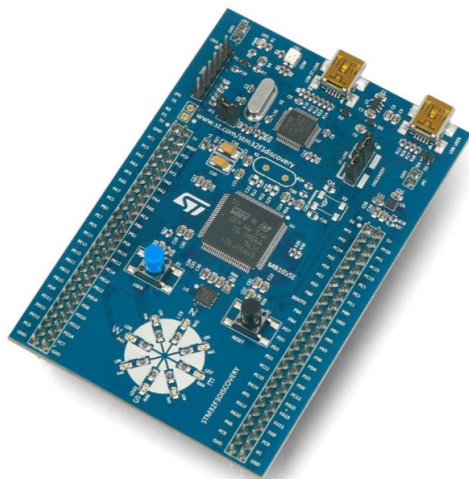
**Napęd:** 2x Koło + silnik 65x26mm 5V z przekładnią.



**Zasilanie:** 3x Ogniwo 18650 Li-Ion INR18650-F1HR 3350mAh, koszyk na 3 akumulatory typu 18650 oraz przetwornica napięcia LM2596HVS



**Sterowanie:** Mikrokontroler STM32F303VCT6 (Discovery), sterownik silnika krokowego 2 DC L293D mini mostek H





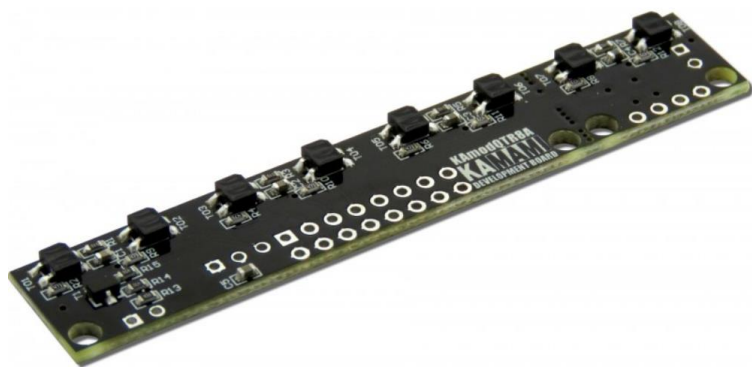
robota

Oliwier Bogdański, Kacper Szponar

**Elementy montażowe:** Śrubki o rozmiarze m3, przewody połączeniowe.



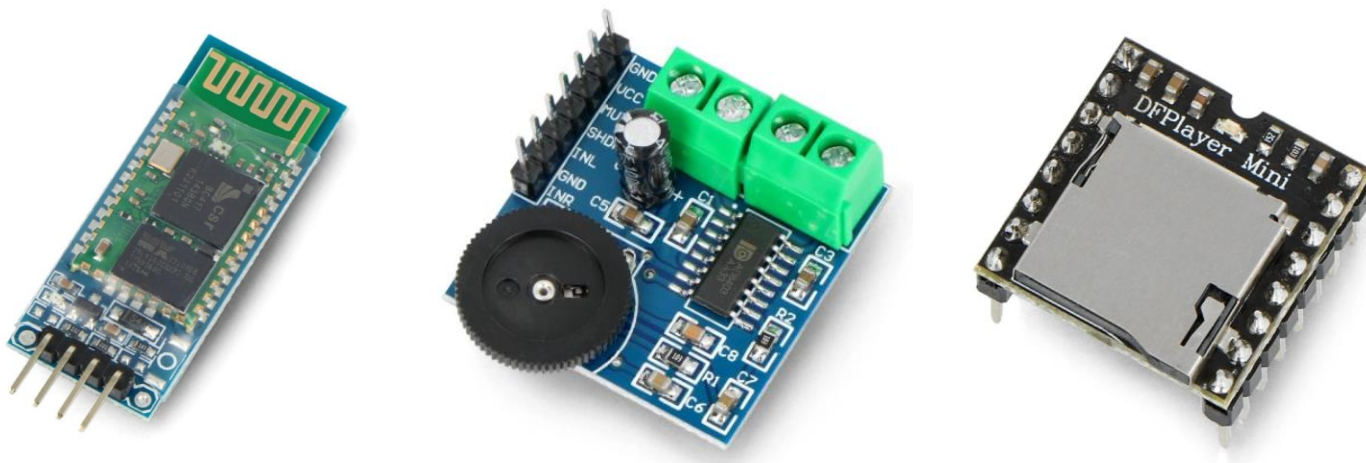
**Czujniki:** KAmoQTR8A - moduł z ośmioma czujnikami odbiciowymi KTIR0711S (wykrycie linii), Ultradźwiękowy czujnik odległości HC-SR04 (wykrycie przeszkody).



## robota

Oliwier Bogdański, Kacper Szponar

**Komunikacja:** Moduł Bluetooth HC-05, DFPlayer mini odtwarzacz MP3 z gniazdem microSD oraz wzmacniacz PAM8403 wzmacniający jego sygnał audio



### 3. Mechanika robota

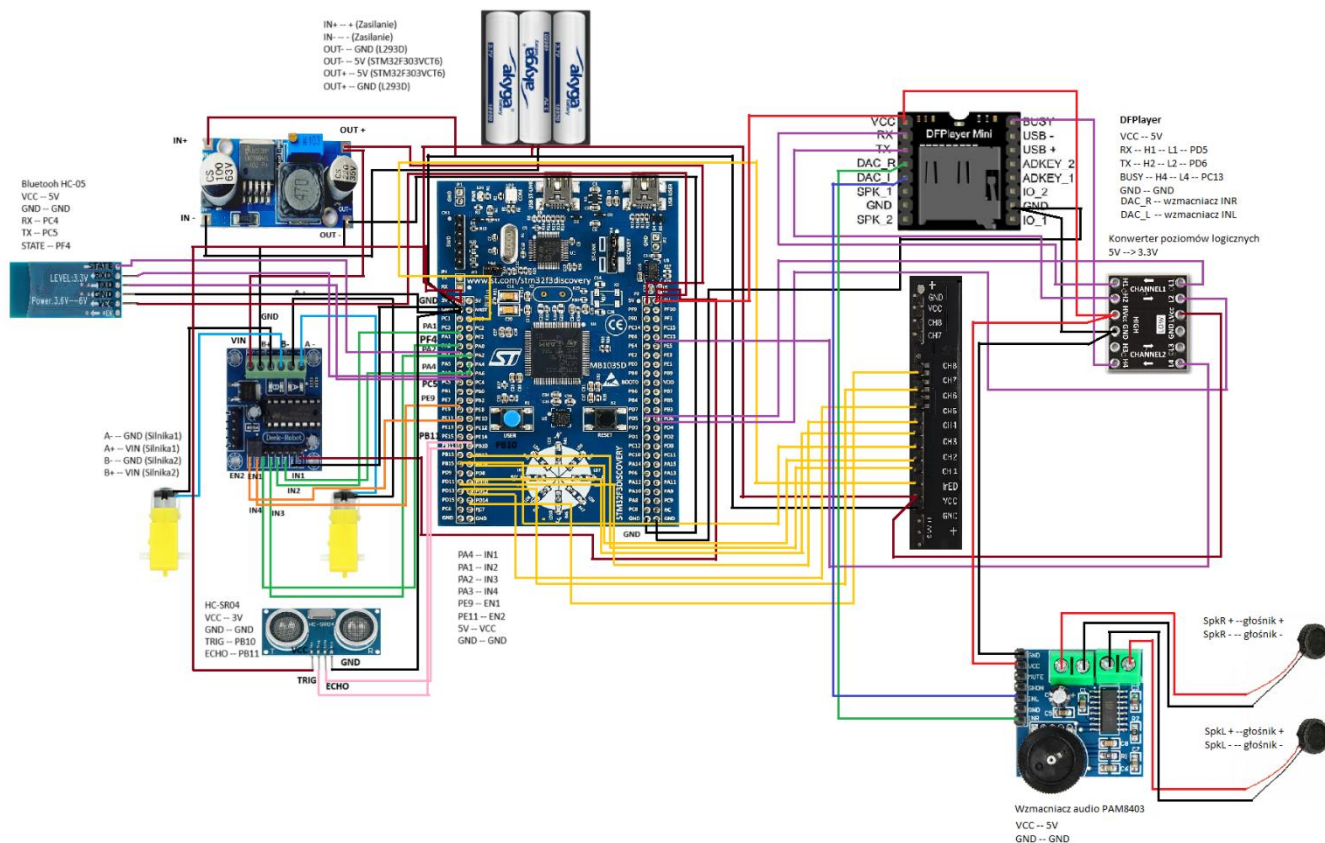
Mechanika robota oparta jest na podwoziu typu Chassis Rectangle 2WD, z dwukołowym napędem oraz z jednym dodatkowym kołem obrotowym. Napęd realizują 2 silniki DC 5V z wbudowaną przekładnią z kołami o wymiarach 65 x 26 mm. Elementy mechaniczne takie jak: koszyk na ogniwa, koło obrotowe, przetwornica, sterownik silników oraz same silniki zostały zamontowane za pomocą śrubek montażowych o rozmiarze m3, podzespoły zostały zamontowane w taki sposób, aby rozkład masy był jak najbardziej równomierny oraz aby zachować stabilność robota. Do połączenia wszystkich elementów zastosowaliśmy przewody połączeniowe typu jumper oraz przewody które zostały przylutowane, które zostały tak poprowadzone, aby nie kolidowały z innymi elementami.



robota

Oliwier Bogdański, Kacper Szponar

#### 4. Schemat elektroniczny robota



robotą

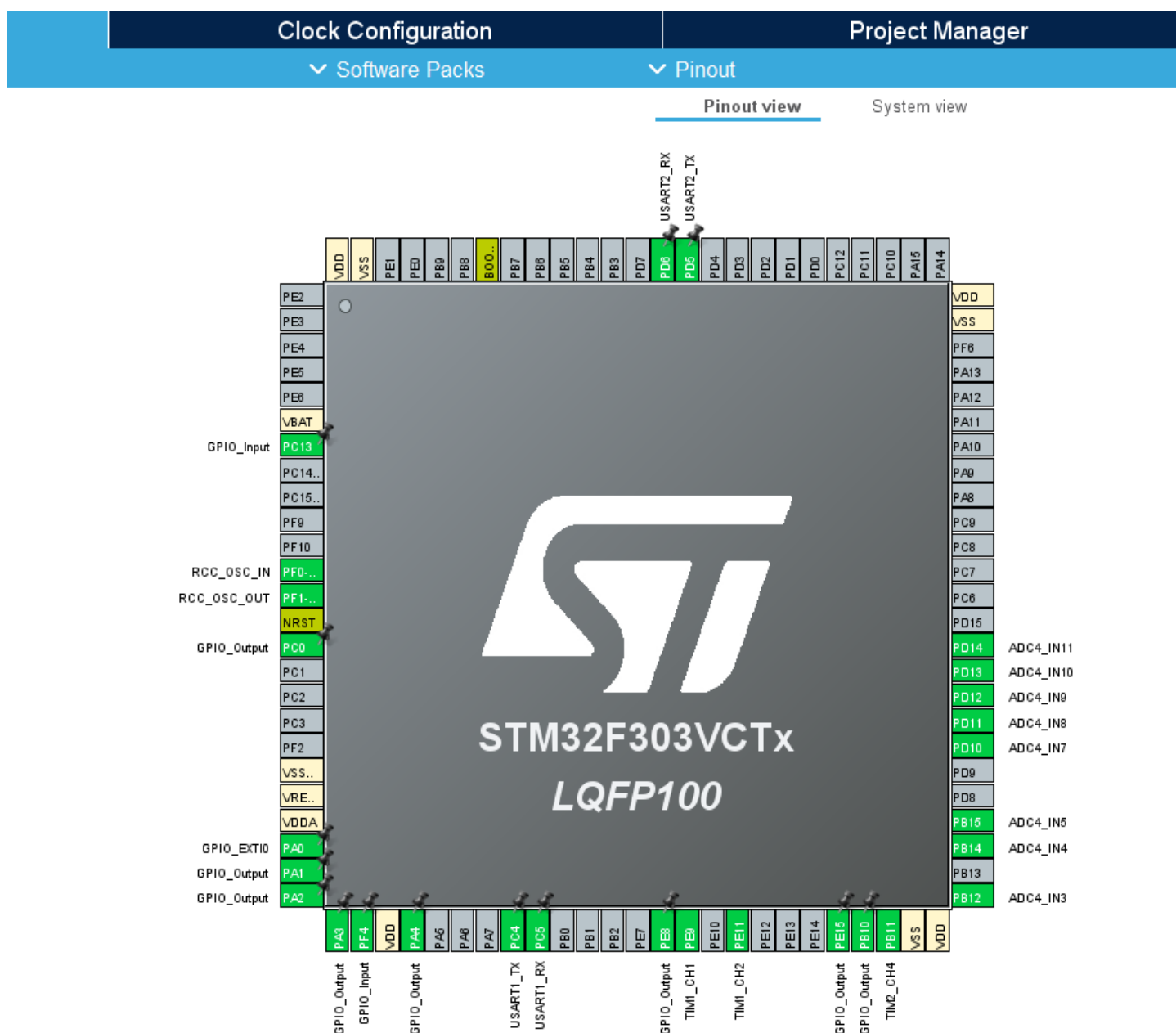
Oliwier Bogdański, Kacper Szponar

## 5. Oprogramowanie sterujące

// Najważniejsze fragmenty kodu źródłowego wraz z opisem

## Konfiguracja ioc

Pinout

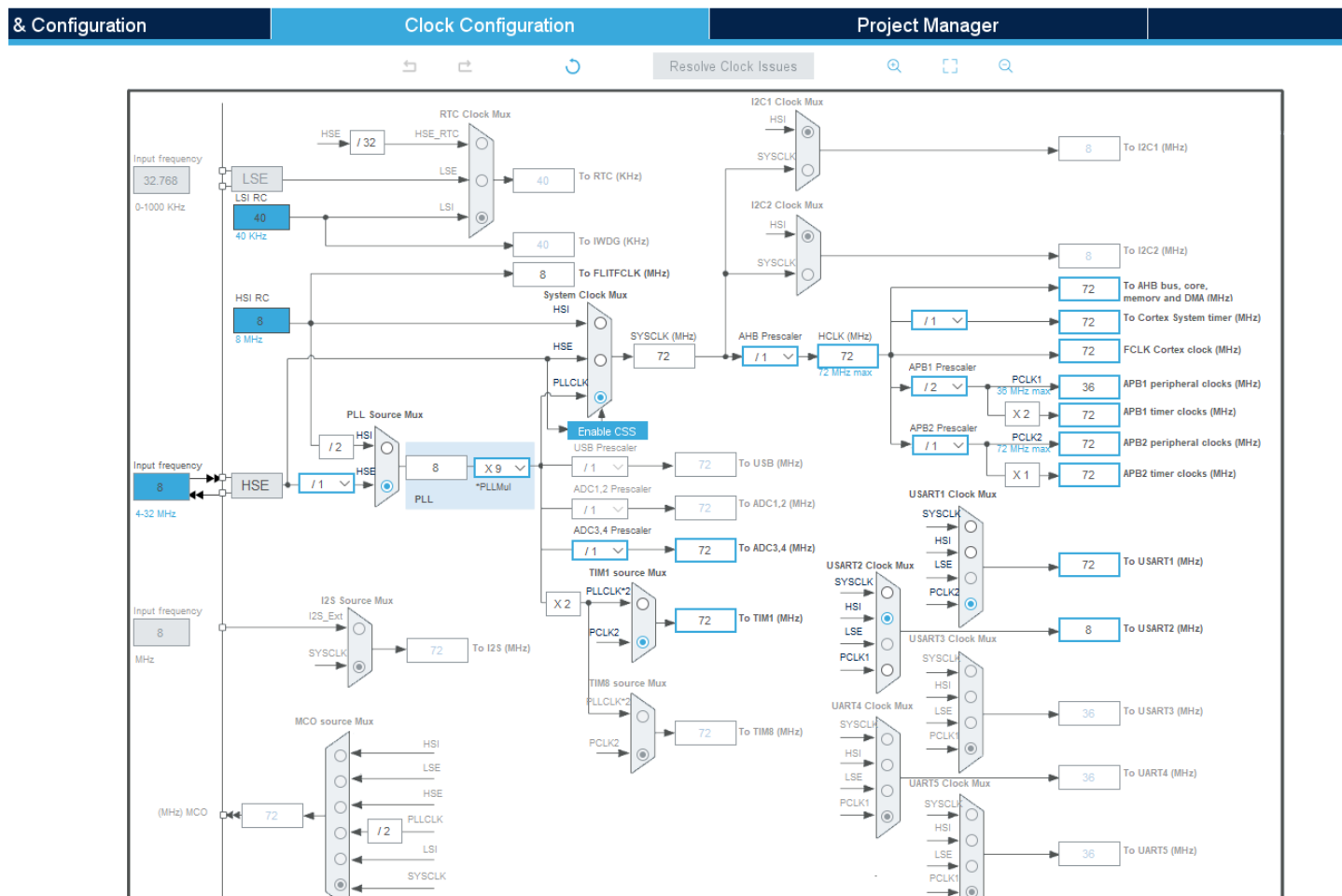




robota

Oliwier Bogdański, Kacper Szponar

CLOCK





robota

Oliwier Bogdański, Kacper Szponar

NVIC

CategoriesA-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

Multimedia

NVIC mode and configuration

Configuration

NVICCode generation

Priority Group...☐ Sort by Preemption Priority and Sub Priority☐

Search⌕Showavailable interrupts

NVIC Interrupt Table	Enabled	Pr
Non maskable interrupt	<input checked="" type="checkbox"/>	0
Hard fault interrupt	<input checked="" type="checkbox"/>	0
Memory management fault	<input checked="" type="checkbox"/>	0
Pre-fetch fault, memory access fault	<input checked="" type="checkbox"/>	0
Undefined instruction or illegal state	<input checked="" type="checkbox"/>	0
System service call via SWI instruction	<input checked="" type="checkbox"/>	0
Debug monitor	<input checked="" type="checkbox"/>	0
Pendable request for system service	<input checked="" type="checkbox"/>	0
Time base: System tick timer	<input checked="" type="checkbox"/>	15
PVD interrupt through EXTI line16	<input type="checkbox"/>	0
Flash global interrupt	<input type="checkbox"/>	0
RCC global interrupt	<input type="checkbox"/>	0
EXTI line0 interrupt	<input checked="" type="checkbox"/>	0
TIM1 break and TIM15 interrupts	<input type="checkbox"/>	0
TIM1 update and TIM16 interrupts	<input type="checkbox"/>	0
TIM1 trigger, commutation and TIM17 interrupts	<input type="checkbox"/>	0
TIM1 capture compare interrupt	<input type="checkbox"/>	0
TIM2 global interrupt	<input checked="" type="checkbox"/>	0
USART1 global interrupt / USART1 wake-up interrupt through...	<input checked="" type="checkbox"/>	0
USART2 global interrupt / USART2 wake-up interrupt through...	<input checked="" type="checkbox"/>	0
DMA2 channel2 global interrupt	<input checked="" type="checkbox"/>	0
ADC4 interrupt	<input type="checkbox"/>	0
Floating point unit interrupt	<input type="checkbox"/>	0





robota

Oliwier Bogdański, Kacper Szponar

## ANALOG ADC4

ADC4 Mode and Configuration

Mode

IN1	Disable	▼
IN2	Disable	▼
IN3	IN3 Single-ended	▼
IN4	IN4 Single-ended	▼
IN5	IN5 Single-ended	▼
IN6	Disable	▼
IN7	IN7 Single-ended	▼
IN8	IN8 Single-ended	▼
IN9	IN9 Single-ended	▼
IN10	IN10 Single-ended	▼
IN11	IN11 Single-ended	▼
IN12	Disable	▼

Configuration

Reset Configuration

✓ NVIC Settings

✓ DMA Settings

✓ GPIO Settings

✓ Parameter Settings

✓ User Constants

Configure the below parameters :

Search (Ctrl+F)

◀

▶

i

▼ ADCs\_Common\_Settings

ModeIndependent mode

▼ ADC\_Settings

Clock PrescalerADC Asynchronous clock mode

ResolutionADC 12-bit resolution

Data AlignmentRight alignment

Scan Conversion ModeEnabled

Continuous Conversion ModeEnabled

Discontinuous Conversion ModeDisabled



robota

Oliwier Bogdański, Kacper Szponar

✓ NVIC Settings

✓ DMA Settings

✓ GPIO Settings

✓ Parameter Settings

✓ User Constants

Configure the below parameters :

Search (Ctrl+F)

◀

▶

i

✓ ADCs\_Common\_Settings

ModeIndependent mode

✓ ADC\_Settings

Clock PrescalerADC Asynchronous clock mode

ResolutionADC 12-bit resolution

Data AlignmentRight alignment

Scan Conversion ModeEnabled

Continuous Conversion ModeEnabled

Discontinuous Conversion ModeDisabled

DMA Continuous RequestsEnabled

End Of Conversion SelectionEnd of single conversion

Overrun behaviourOverrun data overwritten

Low Power Auto WaitDisabled

✓ ADC\_Regular\_ConversionMode

Enable Regular ConversionsEnable

Number Of Conversion8

External Trigger Conversion S...Regular Conversion launched by software

External Trigger Conversion E...None

SequencerNbRanks1

✓ Rank1

ChannelChannel 11

Sampling Time61.5 Cycles

Offset NumberNo offset

Offset0

✓ Rank2

ChannelChannel 10

Sampling Time61.5 Cycles

Offset NumberNo offset



robota

Oliwier Bogdański, Kacper Szponar

Parameter Settings

User Constants

Configure the below parameters :

Search (Ctrl+F)

	Offset	0
▼	Rank	3
	Channel	Channel 9
	Sampling Time	61.5 Cycles
	Offset Number	No offset
	Offset	0
▼	Rank	4
	Channel	Channel 8
	Sampling Time	61.5 Cycles
	Offset Number	No offset
	Offset	0
▼	Rank	5
	Channel	Channel 7
	Sampling Time	61.5 Cycles
	Offset Number	No offset
	Offset	0
▼	Rank	6
	Channel	Channel 5
	Sampling Time	61.5 Cycles
	Offset Number	No offset
	Offset	0
▼	Rank	7
	Channel	Channel 4
	Sampling Time	61.5 Cycles
	Offset Number	No offset
	Offset	0
▼	Rank	8
	Channel	Channel 3
	Sampling Time	61.5 Cycles
	Offset Number	No offset



robota

Oliwier Bogdański, Kacper Szponar

## DMA

Configuration

Reset Configuration

Parameter Settings

User Constants

NVIC Settings

DMA Settings

GPIO Settings

DMA Request	Channel	Direction	Priority
ADC4	ADC4	Peripheral To Memory	Medium

Add

Delete

DMA Request Settings

Mode

Circular

Increment Address

☐

Data Width

Half Word

Peripheral

☐

Memory

☒

Half Word



robota

Oliwier Bogdański, Kacper Szponar

TIM1

Software Packs

Project

TIM1 Mode and Configuration

Mode

Slave Mode

Trigger Source

Clock Source

Channel1

Channel2

Channel3

Configuration

Reset Configuration

☒ Parameter Settings ☒ User Constants ☒ NVIC Settings ☒ DMA Settings ☒ GPIO Settings

Configure the below parameters :

Search (Ctrl+F)

Counter Settings

Prescaler (PSC - 16 bits value) 71

Counter Mode Up

Counter Period (AutoReload Register - 16 bits value ) 999

Internal Clock Division (CKD) No Division

Repetition Counter (RCR - 16 bits value) 0

auto-reload preload Disable

Trigger Output (TRGO) Parameters

Master/Slave Mode (MSM bit) Disable (Trigger input effect not delayed)

Trigger Event Selection TRGO Reset (UG bit from TIMx\_EGR)

Trigger Event Selection TRGO2 Reset (UG bit from TIMx\_EGR)

Break And Dead Time management - BRK Configuration

BRK State Disable





robota

Oliwier Bogdański, Kacper Szponar

TIM2

TIM2 Mode and Configuration

Mode

Slave Mode

Disable

▼

Trigger Source

Disable

▼

Clock Source

Internal Clock

▼

Channel1

Disable

▼

Channel2

Disable

▼

Channel3

Disable

▼

Channel4

Input Capture direct mode

▼

Combined Channels

Disable

▼

☐ ETR IO as Clearing Source

Configuration

Reset Configuration

✔ Parameter Settings

✔ User Constants

✔ NVIC Settings

✔ DMA Settings

✔ GPIO Settings

Configure the below parameters :

◀

▶

❗

▼ Counter Settings

Prescaler (PSC - 16 bits value)

0

Counter Mode

Up

Counter Period (AutoReload Register - 32 bits value )

0xffffffff

Internal Clock Division (CKD)

No Division

auto-reload preload

Disable

▼ Trigger Output (TRGO) Parameters

Master/Slave Mode (MSM bit)

Disable (Trigger input effect not delayed)

Trigger Event Selection TRGO

Reset (UG bit from TIMx\_EGR)

▼ Input Capture Channel 4

Polarity Selection

Rising Edge

IC Selection

Direct

Prescaler Division Ratio

No division

Input Filter (4 bits value)

0

Strona 14/47



robota

Oliwier Bogdański, Kacper Szponar

## USART1

The screenshot displays the STM32CubeMX configuration tool. On the left, a sidebar lists various components under 'Connectivity', with 'USART1' selected and highlighted in blue. The main window is titled 'USART1 Mode and Configuration'. It is divided into two main sections: 'Mode' and 'Configuration'. The 'Mode' section includes a dropdown for 'Mode' set to 'Asynchronous', a dropdown for 'Hardware Flow Control (RS232)' set to 'Disable', and an unchecked checkbox for 'Hardware Flow Control (RS485)'. The 'Configuration' section has a 'Reset Configuration' button and a row of tabs: 'Parameter Settings' (selected), 'User Constants', 'NVIC Settings', 'DMA Settings', and 'GPIO Settings'. Below the tabs, a search bar is present. The configuration parameters are organized into three expandable sections: '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), and 'Advanced Features' (Auto Baudrate: Disable, TX Pin Active Level Inversion: Disable, RX Pin Active Level Inversion: Disable, Data Inversion: Disable, TX and RX Pins Swapping: Disable, Overrun: Enable, DMA on RX Error: Enable, MSB First: Disable).



robota

Oliwier Bogdański, Kacper Szponar

USART2

🔍

Categories

A->Z

⚠️ 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

USART2 Mode and Configuration

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)

⏪

⏩

ℹ️

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 Inver... Disable

Data Inversion Disable

TX and RX Pins Swapping Disable

Overrun Enable

DMA on RX Error Enable

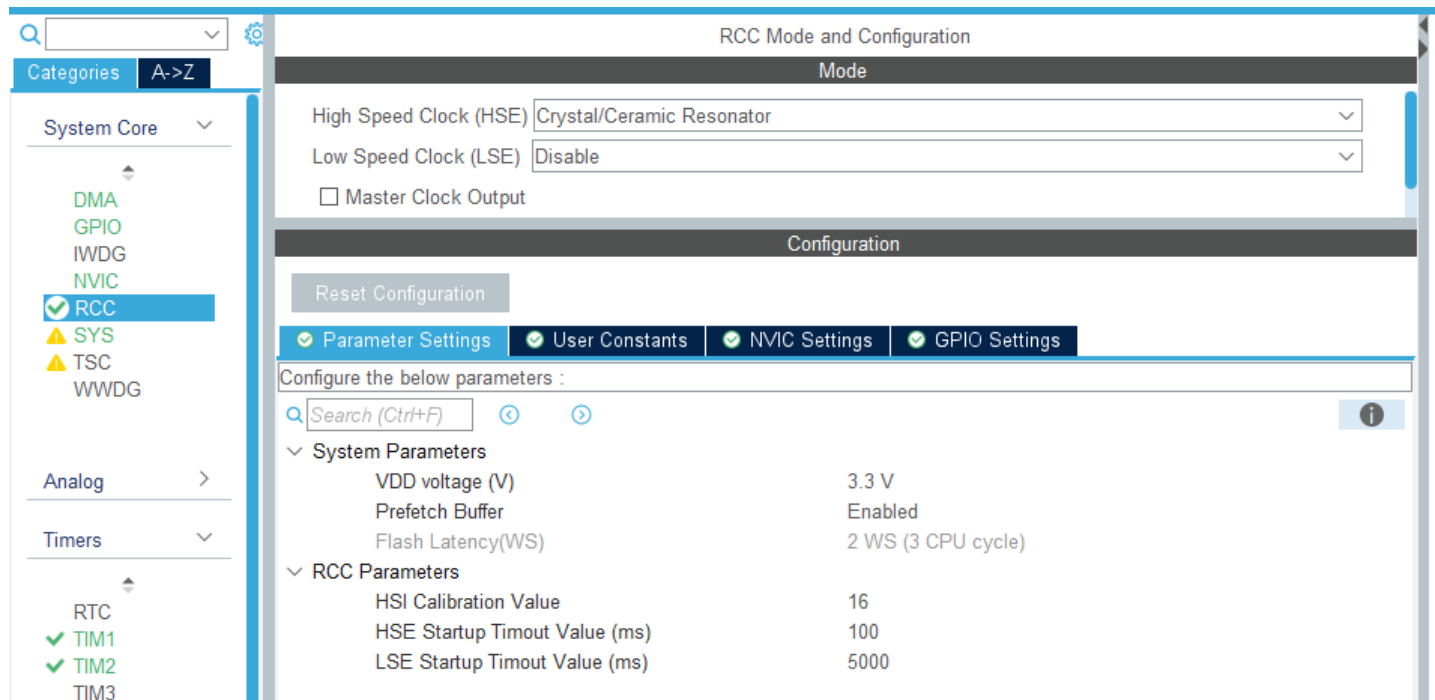
MSB First Disable



robota

Oliwier Bogdański, Kacper Szponar

## RCC



## Kod Programu

### Main.c

```
17  /*  
18  /* USER CODE END Header */  
19  /* Includes -----*/  
20  #include "main.h"  
21  
22  /* Private includes -----*/  
23  /* USER CODE BEGIN Includes */  
24  #include "line_follower.h"  
25  #include "dfplayer_mini.h"  
26  #include <string.h> // Potrzebne dla strlen  
27  #include <stdio.h>  
28  #include <ctype.h>  
29  /* USER CODE END Includes */  
30  
31  /* Private typedef -----*/  
32  /* USER CODE BEGIN PTD */  
33  
34  /* USER CODE END PTD */  
35  
36  /* Private define -----*/  
37  /* USER CODE BEGIN PD */  
38  
39  // Typ enum dla trybów pracy robota  
40  typedef enum {  
41      ROBOT_MODE_STOPPED,  
42      ROBOT_MODE_LINE_FOLLOWER,  
43      ROBOT_MODE_BLUETOOTH_MANUAL  
44  } RobotOperatingMode;  
45  
46  #define ADC_CHANNELS 8 // Definicja liczby kanałów ADC używanych do odczytu czujników linii  
47  #define BT_COMMAND_BUFFER_SIZE 32 // Maksymalna długość komendy BT  
48  
49  uint16_t adc_buffer[ADC_CHANNELS]; // Bufor przechowujący odczyty z czujników linii (ADC)  
50  
51  volatile uint8_t robot_running = 0; // Zmienna globalna ustawiana w przerwaniu po naciśnięciu przycisku użytkownika (USER button).  
52  
53  volatile uint32_t echo_start = 0; // Czas (w tickach timera) rozpoczęcia odbieranego echa  
54  volatile uint32_t echo_end = 0; // Czas (w tickach timera) zakończenia odbieranego echa  
55  volatile uint8_t echo_captured = 0; // Flaga statusu przechwytywania echa (0: nic, 1: start, 2: koniec).  
56  
57  static char bt_command_buffer[BT_COMMAND_BUFFER_SIZE]; // Bufor na komendy z Bluetooth  
58  static volatile uint8_t bt_cmd_buffer_idx = 0; // Indeks w buforze komend BT  
59  
60  // Początkowe wartości PWM dla sterowania Bluetooth  
61  const uint16_t INITIAL_BT_PWM_SPEED_LEFT = 999; // Początkowa prędkość dla lewego silnika przy sterowaniu BT  
62  const uint16_t INITIAL_BT_PWM_SPEED_RIGHT = 900; // Początkowa prędkość dla prawego silnika przy sterowaniu BT  
63  /* USER CODE END PD */  
64
```



robota

Oliwier Bogdański, Kacper Szponar

```
80 /* USER CODE BEGIN PV */
81 volatile RobotOperatingMode current_robot_mode = ROBOT_MODE_STOPPED; // Domyślny tryb robota
82 static uint8_t dfp_current_volume = 12; // Domyślna głośność (0-30)
83 static uint8_t dfp_is_commanded_to_play = 0; // Czy użytkownik zażądał odtwarzania
84 static uint8_t dfp_is_user_paused = 0; // Czy użytkownik zapauzował
85
86 // Zmienne przechowujące aktualne docelowe prędkości PWM dla sterowania BT
87 static uint16_t current_bt_pwm_left = INITIAL_BT_PWM_SPEED_LEFT;
88 static uint16_t current_bt_pwm_right = INITIAL_BT_PWM_SPEED_RIGHT;
89
90 // Stała różnica prędkości między lewym a prawym silnikiem, obliczana na podstawie wartości początkowych
91 static const int16_t pwm_speed_diff = INITIAL_BT_PWM_SPEED_LEFT - INITIAL_BT_PWM_SPEED_RIGHT;
92 /* USER CODE END PV */
93
94 /* Private function prototypes -----*/
95 void SystemClock_Config(void);
96 static void MX_GPIO_Init(void);
97 static void MX_DMA_Init(void);
98 static void MX_TIM1_Init(void);
99 static void MX_TIM2_Init(void);
100 static void MX_USART1_UART_Init(void);
101 static void MX_ADC4_Init(void);
102 static void MX_USART2_UART_Init(void);
103 /* USER CODE BEGIN PFP */
104
105 // Prototypy funkcji, które mogą być używane przez inne moduły (np. line_follower)
106 void robot_drive(uint16_t pwm_right, uint8_t dir_right, uint16_t pwm_left, uint8_t dir_left);
107 void robot_stop(void);
108 void UART_SendString(char *str);
109 float HCSR04_ReadDistance(void);
110 void HCSR04_Trigger(void);
111 uint8_t HC05_State(void);
112
113 void ProcessBluetoothCommand(char* command_str);
114 /* USER CODE END PFP */
115
```





robota

Oliwier Bogdański, Kacper Szponar

```
119 //Funkcja do uruchamiania robota w żądanym kierunku i prędkością za pomocą parametrów
120
121 void robot_drive(uint16_t pwm_right, uint8_t dir_right,
122                 uint16_t pwm_left, uint8_t dir_left)
123 {
124     __HAL_TIM_SET_COMPARE(&htim1, TIM_CHANNEL_1, pwm_right); // Ustawia wypełnienie PWM na kanale 1 (prawy silnik) - kontrola prędkości
125     __HAL_TIM_SET_COMPARE(&htim1, TIM_CHANNEL_2, pwm_left); // Ustawia wypełnienie PWM na kanale 2 (lewy silnik) - kontrola prędkości
126     // ustawianie odpowiednich stanów logicznych na wyjściach GPIO
127     // prawy silnik
128     if (dir_right) {
129         HAL_GPIO_WritePin(GPIOA, GPIO_PIN_4, GPIO_PIN_SET); // IN1
130         HAL_GPIO_WritePin(GPIOA, GPIO_PIN_1, GPIO_PIN_RESET); // IN2
131     } else {
132         HAL_GPIO_WritePin(GPIOA, GPIO_PIN_4, GPIO_PIN_RESET); // IN1
133         HAL_GPIO_WritePin(GPIOA, GPIO_PIN_1, GPIO_PIN_SET); // IN2
134     }
135
136     // lewy silnik
137     if (dir_left) {
138         HAL_GPIO_WritePin(GPIOA, GPIO_PIN_2, GPIO_PIN_SET); // IN3
139         HAL_GPIO_WritePin(GPIOA, GPIO_PIN_3, GPIO_PIN_RESET); // IN4
140     } else {
141         HAL_GPIO_WritePin(GPIOA, GPIO_PIN_2, GPIO_PIN_RESET); // IN3
142         HAL_GPIO_WritePin(GPIOA, GPIO_PIN_3, GPIO_PIN_SET); // IN4
143     }
144 }
145
146 // Funkcja zatrzymująca ruch robota. Ustawia wszystkie wyjścia GPIO NA 0 oraz wypełnienie PWM na 0 na obu silnikach
147 void robot_stop()
148 {
149     HAL_GPIO_WritePin(GPIOA, GPIO_PIN_4, GPIO_PIN_RESET);
150     HAL_GPIO_WritePin(GPIOA, GPIO_PIN_1, GPIO_PIN_RESET);
151     HAL_GPIO_WritePin(GPIOA, GPIO_PIN_2, GPIO_PIN_RESET);
152     HAL_GPIO_WritePin(GPIOA, GPIO_PIN_3, GPIO_PIN_RESET);
153     __HAL_TIM_SET_COMPARE(&htim1, TIM_CHANNEL_1, 0);
154     __HAL_TIM_SET_COMPARE(&htim1, TIM_CHANNEL_2, 0);
155 }
156
157 // Funkcja Generuje krótki (10µs) impuls na pinie TRIG czujnika HC-SR04, aby zainicjować pomiar odległości
158 void HCSR04_Trigger(void)
159 {
160     // Ustawia pin PB10 (TRIG) w stan wysoki, czeka 10 mikrosekund, a następnie ustawia go w stan niski.
161     HAL_GPIO_WritePin(GPIOB, GPIO_PIN_10, GPIO_PIN_SET);
162     HAL_Delay(0.01); // 10 us za pomocą HAL Delay. Do zmiany w przyszłości aby liczyło czas za pomocą timera.
163     HAL_GPIO_WritePin(GPIOB, GPIO_PIN_10, GPIO_PIN_RESET);
164 }
165
```



robota

Oliwier Bogdański, Kacper Szponar

```
167 // Funkcja mierząca odległość za pomocą czujnika HC-SR04
168 // Inicjuje pomiar, czeka na impuls echa, oblicza czas jego trwania i przelicza na odległość w cm
169 // Używa timera TIM2 w trybie Input Capture do pomiaru czasu trwania impulsu echa
170 float HCSR04_ReadDistance(void)
171 {
172     echo_captured = 0; // Reset flagi statusu przechwytywania echa
173
174     // Uruchamia timer TIM2 w trybie Input Capture na kanale 4 (oczekiwanie na zbocze narastające)
175     HAL_TIM_IC_Start_IT(&htim2, TIM_CHANNEL_4);
176
177     HCSR04_Trigger(); // Wywołanie funkcji wysyłającej impuls
178
179     uint32_t timeout = HAL_GetTick() + 100; // timeout na 100ms
180     while (echo_captured < 2) // Czekaj, aż zostaną przechwycone oba zbocza impulsu echa (start i end)
181     {
182         if (HAL_GetTick() > timeout)
183         {
184             return -1.0f; // wystąpił timeout (brak echa w ciągu 100ms), funkcja zwraca błąd -1.0f
185         }
186     }
187     // Zatrzymanie timera
188     HAL_TIM_IC_Stop_IT(&htim2, TIM_CHANNEL_4);
189
190     uint32_t ticks; // Zmienna na czas trwania impulsu echa w tickach timera
191     if (echo_end >= echo_start) // Normalny przypadek
192         ticks = echo_end - echo_start;
193     else // Przypadek przepełnienia licznika timera
194         ticks = (0xFFFFFFFF - echo_start + echo_end);
195
196     float distance = ticks * 0.000239463f;
197     // Obliczony współczynnik dla zegara systemowego 72MHz i prędkości dźwięku ok 343m/s
198     // Daje on wystarczającą dokładność dla wykrywania przeszkód podczas jazdy oraz
199     // mierzenia dystansu w zakresie pracy czujnika
200
201     return distance; // Zwrócenie obliczonej odległości
202 }
203
204 // Funkcja pomocnicza do wysyłająca wiadomości poprzez UART
205 // Jako parametr przyjmuje wskaźnik do ciągu znaków do wysłania
206 void UART_SendString(char *str) {
207     HAL_UART_Transmit(&huart1, (uint8_t*)str, strlen(str), HAL_MAX_DELAY);
208 }
209
210
211 // Funkcja sprawdzająca stan połączenia Bluetooth
212 uint8_t HC05_State(void) {
213     // Zwraca 1 jeśli połączony (HC-05 wysyła HIGH na pinie State)
214     return HAL_GPIO_ReadPin(GPIOF, GPIO_PIN_4);
215 }
216
```



robota

Oliwier Bogdański, Kacper Szponar

```
217
218 // Funkcja przetwarzająca komendy z Bluetooth sterując ruchem robota lub zmieniając jego tryb pracy.
219 // Wywoływana po odebraniu pełnej komendy zakończonej znakiem nowej linii.
220 void ProcessBluetoothCommand(char* command_str) {
221     char msg[70]; // Bufor na wiadomości dla UART
222     int value; // Zmienna do przechowywania wartości liczbowych z komend (np. głośność, numer utworu, procent prędkości)
223
224     // Wysłanie potwierdzenia odebrania komendy (debugowanie)
225     // snprintf(msg, sizeof(msg), "BT Odebrana komenda: [%s]\r\n", command_str);
226     // UART_SendString(msg);
227
228     if (strlen(command_str) == 0) { // Ignoruje puste komendy
229         return;
230     }
231
232     char command_char = tolower(command_str[0]); // Pobiera pierwszy znak komendy i zamienia na małą literę
233
234     switch (command_char) {
235     case 'f': // Przód
236         if (strlen(command_str) == 1) {
237             if (current_robot_mode == ROBOT_MODE_BLUETOOTH_MANUAL) {
238                 robot_drive(current_bt_pwm_right, 1, current_bt_pwm_left, 1);
239                 UART_SendString("BT CMD: FWD\r\n");
240             } else {
241                 UART_SendString("BT Info: Tryb manualny nie aktywny\r\n");
242             }
243         } else {
244             snprintf(msg, sizeof(msg), "BT CMD: Nieznana komenda '%s'\r\n", command_str);
245             UART_SendString(msg);
246         }
247         break;
248     case 'b': // Tył
249         if (strlen(command_str) == 1) {
250             if (current_robot_mode == ROBOT_MODE_BLUETOOTH_MANUAL) {
251                 robot_drive(current_bt_pwm_right, 0, current_bt_pwm_left, 0);
252                 UART_SendString("BT CMD: BCK\r\n");
253             } else {
254                 UART_SendString("BT Info: Tryb manualny nie aktywny\r\n");
255             }
256         } else {
257             snprintf(msg, sizeof(msg), "BT CMD: Nieznana komenda '%s'\r\n", command_str);
258             UART_SendString(msg);
259         }
260         break;
261     case 'l': // Lewo
262         if (strlen(command_str) == 1) {
263             if (current_robot_mode == ROBOT_MODE_BLUETOOTH_MANUAL) {
264                 robot_drive(current_bt_pwm_right, 1, current_bt_pwm_left, 0);
265                 UART_SendString("BT CMD: LEFT\r\n");
266             } else {
267                 UART_SendString("BT Info: Tryb manualny nie aktywny\r\n");
268             }
269         } else {
270             snprintf(msg, sizeof(msg), "BT CMD: Nieznana komenda '%s'\r\n", command_str);
271             UART_SendString(msg);
272         }
273         break;
274     }
```



robota

Oliwier Bogdański, Kacper Szponar

```
259     }
260     break;
261 case 'l': // Lewo
262     if (strlen(command_str) == 1) {
263         if (current_robot_mode == ROBOT_MODE_BLUETOOTH_MANUAL) {
264             robot_drive(current_bt_pwm_right, 1, current_bt_pwm_left, 0);
265             UART_SendString("BT CMD: LEFT\r\n");
266         } else {
267             UART_SendString("BT Info: Tryb manualny nie aktywny\r\n");
268         }
269     } else {
270         snprintf(msg, sizeof(msg), "BT CMD: Nieznana komenda '%s'\r\n", command_str);
271         UART_SendString(msg);
272     }
273     break;
274 case 'r': // Prawo
275     if (strlen(command_str) == 1) {
276         if (current_robot_mode == ROBOT_MODE_BLUETOOTH_MANUAL) {
277             robot_drive(current_bt_pwm_right, 0, current_bt_pwm_left, 1);
278             UART_SendString("BT CMD: RIGHT\r\n");
279         } else {
280             UART_SendString("BT Info: Tryb manualny nie aktywny\r\n");
281         }
282     } else {
283         snprintf(msg, sizeof(msg), "BT CMD: Nieznana komenda '%s'\r\n", command_str);
284         UART_SendString(msg);
285     }
286     break;
287 case 's': // Stop
288     if (strlen(command_str) == 1) {
289         robot_running = 0; // flaga na 0
290         robot_stop();
291         current_robot_mode = ROBOT_MODE_BLUETOOTH_MANUAL; // Po stop pozostaje w trybie manualnym aby płynnej sterować aplikacją w telefonie
292         line_follower_reset_state();
293         UART_SendString("BT CMD: STOP\r\n");
294     } else {
295         snprintf(msg, sizeof(msg), "BT CMD: Nieznana komenda '%s'\r\n", command_str);
296         UART_SendString(msg);
297     }
298     break;
299
300 case 'a': // Tryb autonomiczny - śledzenie linii
301     if (strlen(command_str) == 1) {
302         if (current_robot_mode != ROBOT_MODE_LINE_FOLLOWER || !robot_running) { // Jeśli nie był już w LF i nie jechał
303             robot_stop(); // Zatrzymanie, jeśli np. jedzie w trybie BT
304         }
305         current_robot_mode = ROBOT_MODE_LINE_FOLLOWER;
306         robot_running = 1; // Odrazu jedzie (zmienna z PA0 ustawiona na 1)
307         line_follower_reset_state();
308         UART_SendString("BT CMD: Tryb -> Śledzenie linii\r\n");
309     } else {
```



robota

Oliwier Bogdański, Kacper Szponar

```
299
300 case 'a': // Tryb autonomiczny - śledzenie linii
301     if (strlen(command_str) == 1) {
302         if (current_robot_mode != ROBOT_MODE_LINE_FOLLOWER || !robot_running) { // Jeśli nie był już w LF i nie jechał
303             robot_stop(); // Zatrzymanie, jeśli np. jedzie w trybie BT
304         }
305         current_robot_mode = ROBOT_MODE_LINE_FOLLOWER;
306         robot_running = 1; // Odrazu jedzie (zmienna z PA0 ustawiona na 1)
307         line_follower_reset_state();
308         UART_SendString("BT CMD: Tryb -> Śledzenie linii\r\n");
309     } else {
310         snprintf(msg, sizeof(msg), "BT CMD: Nieznana komenda '%s'\r\n", command_str);
311         UART_SendString(msg);
312     }
313     break;
314
315 case 'm': // Tryb manualny - sterowanie Bluetooth
316     if (strlen(command_str) == 1) {
317         if (robot_running && current_robot_mode == ROBOT_MODE_LINE_FOLLOWER) {
318             robot_stop(); // Zatrzymanie jeśli jedzie
319         }
320         current_robot_mode = ROBOT_MODE_BLUETOOTH_MANUAL;
321         robot_running = 0; // W trybie manualnym, 'robot_running' nie kontroluje robotem bezpośrednio
322         line_follower_reset_state();
323         UART_SendString("BT CMD: Tryb -> Sterowanie ręczne\r\n");
324     } else {
325         snprintf(msg, sizeof(msg), "BT CMD: Nieznana komenda '%s'\r\n", command_str);
326         UART_SendString(msg);
327     }
328     break;
329
330 // --- Sterowanie DFPlayer Mini ---
331
332
333 case '1': // Odtwórz utwór 1
334     dfplayer_play_track(1);
335     dfp_is_commanded_to_play = 1;
336     dfp_is_user_paused = 0;
337     UART_SendString("BT CMD: Track 1\r\n");
338     break;
339
```





robotą

Oliwier Bogdański, Kacper Szponar

```
341 // Odtwórz utwór T<numer>
342 case 't': // Komenda 'T' z parametrem numeru utworu
343     if (command_char == 't' && strlen(command_str) > 1) {
344         if (sscanf(command_str + 1, "%d", &value) == 1) {
345             if (value > 0 && value <= 255) { // DFPlayer obsługuje do 255 utworów w folderze głównym
346                 dfplayer_play_track((uint16_t)value);
347                 dfp_is_commanded_to_play = 1;
348                 dfp_is_user_paused = 0;
349                 snprintf(msg, sizeof(msg), "BT CMD: Playing Track %d\r\n", value);
350                 UART_SendString(msg);
351             } else {
352                 UART_SendString("BT ERR: Numer utworu poza zakresem (1-255)\r\n");
353             }
354         } else {
355             UART_SendString("BT ERR: Bledny format komendy utworu. Uzyj T<numer_1_255>\r\n");
356         }
357     } else if (command_char == 't' && strlen(command_str) == 1) { // Samo 't' bez parametru
358         UART_SendString("BT ERR: Komenda T wymaga numeru utworu. Uzyj T<numer_1_255>\r\n");
359     } else { // Coś innego zaczynającego się na 't'
360         snprintf(msg, sizeof(msg), "BT CMD: Nieznana komenda '%s'\r\n", command_str);
361         UART_SendString(msg);
362     }
363     break;
364
365 case '+': // Głośniej
366     if (strlen(command_str) == 1) {
367         if (dfp_current_volume < 30) {
368             dfp_current_volume++;
369         }
370         dfplayer_set_volume(dfp_current_volume);
371         snprintf(msg, sizeof(msg), "BT CMD: Glosnosc w gore (%d/30)\r\n", dfp_current_volume);
372         UART_SendString(msg);
373     } else {
374         snprintf(msg, sizeof(msg), "BT CMD: Nieznana komenda '%s'\r\n", command_str);
375         UART_SendString(msg);
376     }
377     break;
378 case '-': // Ciszej
379     if (strlen(command_str) == 1) {
380         if (dfp_current_volume > 0) {
381             dfp_current_volume--;
382         }
383         dfplayer_set_volume(dfp_current_volume);
384         snprintf(msg, sizeof(msg), "BT CMD: Glosnosc w dol (%d/30)\r\n", dfp_current_volume);
385         UART_SendString(msg);
386     } else {
387         snprintf(msg, sizeof(msg), "BT CMD: Nieznana komenda '%s'\r\n", command_str);
388         UART_SendString(msg);
389     }
390     break;
391 case 'n': // Następny utwór
392     if (strlen(command_str) == 1) {
393         dfplayer_next_track();
394     }
```



robota

Oliwier Bogdański, Kacper Szponar

```
391     case 'n': // Następny utwór
392         if (strlen(command_str) == 1) {
393             dfplayer_next_track();
394             dfp_is_commanded_to_play = 1;
395             dfp_is_user_paused = 0;
396             UART_SendString("BT CMD: Następny utwór\r\n");
397         } else {
398             snprintf(msg, sizeof(msg), "BT CMD: Nieznana komenda '%s'\r\n", command_str);
399             UART_SendString(msg);
400         }
401         break;
402     case 'v': // Poprzedni utwór (v) LUB Ustaw głośność (V<poziom>)
403         if (command_char == 'v' && strlen(command_str) == 1) {
404             dfplayer_prev_track();
405             dfp_is_commanded_to_play = 1;
406             dfp_is_user_paused = 0;
407             UART_SendString("BT CMD: Poprzedni utwór\r\n");
408         }
409         // Ustaw głośność V<poziom>
410         else if (command_char == 'v' && strlen(command_str) > 1) { // Komenda 'V' z parametrem głośności
411             if (sscanf(command_str + 1, "%d", &value) == 1) {
412                 if (value >= 0 && value <= 30) {
413                     dfp_current_volume = (uint8_t)value;
414                     dfplayer_set_volume(dfp_current_volume);
415                     snprintf(msg, sizeof(msg), "BT CMD: Glosnosc ustawiona na %d/30\r\n", dfp_current_volume);
416                     UART_SendString(msg);
417                 } else {
418                     UART_SendString("BT ERR: Glosnosc poza zakresem (0-30)\r\n");
419                 }
420             } else {
421                 UART_SendString("BT ERR: Bledny format komendy glosnosci. Uzyj V<0-30>\r\n");
422             }
423         }
424         else {
425             snprintf(msg, sizeof(msg), "BT CMD: Nieznana komenda '%s'\r\n", command_str);
426             UART_SendString(msg);
427         }
428         break;
429     case 'p': // Pauza/Wznów
430         if (strlen(command_str) == 1) {
431             if (dfp_is_commanded_to_play) { // Działa tylko, jeśli muzyka była odtwarzana
432                 if (!dfp_is_user_paused) {
433                     dfplayer_pause();
434                     dfp_is_user_paused = 1;
435                     UART_SendString("BT CMD: Pauza\r\n");
436                 } else {
437                     dfplayer_resume();
438                     dfp_is_user_paused = 0;
439                     UART_SendString("BT CMD: Wznowiono\r\n");
440                 }
441             } else {
442                 UART_SendString("BT Info: Nie odtwarzam muzyki.\r\n");
443             }
444         }
445     }
```



robota

Oliwier Bogdański, Kacper Szponar

```
427     }  
428     break;  
429     case 'p': // Pauza/Wznów  
430         if (strlen(command_str) == 1) {  
431             if (dfp_is_commanded_to_play) { // Działa tylko, jeśli muzyka była odtwarzana  
432                 if (!dfp_is_user_paused) {  
433                     dfplayer_pause();  
434                     dfp_is_user_paused = 1;  
435                     UART_SendString("BT CMD: Pauza\r\n");  
436                 } else {  
437                     dfplayer_resume();  
438                     dfp_is_user_paused = 0;  
439                     UART_SendString("BT CMD: Wznowiono\r\n");  
440                 }  
441             } else {  
442                 UART_SendString("BT Info: Nie odtwarzam muzyki.\r\n");  
443             }  
444         } else {  
445             snprintf(msg, sizeof(msg), "BT CMD: Nieznana komenda '%s'\r\n", command_str);  
446             UART_SendString(msg);  
447         }  
448         break;  
449     case 'x': // Stop odtwarzania  
450         if (strlen(command_str) == 1) {  
451             dfplayer_stop();  
452             dfp_is_commanded_to_play = 0;  
453             dfp_is_user_paused = 0;  
454             UART_SendString("BT CMD: Muzyka stop\r\n");  
455         } else {  
456             snprintf(msg, sizeof(msg), "BT CMD: Nieznana komenda '%s'\r\n", command_str);  
457             UART_SendString(msg);  
458         }  
459         break;  
460  
461     // Ustaw prędkość W<procent_0_100>  
462     case 'w': // Komenda 'W' z parametrem procentu prędkości  
463         if (command_char == 'w' && strlen(command_str) > 1) { // Sprawdzanie czy jest podany parametr  
464             if (sscanf(command_str + 1, "%d", &value) == 1) {  
465                 if (value >= 0 && value <= 100) {  
466                     // Obliczanie docelowego PWM dla lewego silnika (0-999) na podstawie procentu  
467                     uint16_t target_pwm_left = (uint16_t)((value / 100.0f) * 999.0f);  
468  
469                     // Ograniczenie górne PWM do 999  
470                     if (target_pwm_left > 999) target_pwm_left = 999;  
471  
472                     current_bt_pwm_left = target_pwm_left; // Ustawia nowe PWM dla lewego silnika  
473  
474                     // Obliczanie PWM dla prawego silnika używając stałej różnicy  
475                     int16_t calculated_right_pwm = target_pwm_left - pwm_speed_diff;  
476  
477                     // Sprawdzanie czy PWM dla prawego silnika nie jest ujemne i nie przekracza 999  
478                     if (calculated_right_pwm < 0) {  
479                         current_bt_pwm_right = 0;  
480                     }  
481                 }  
482             }  
483         }
```



robota

Oliwier Bogdański, Kacper Szponar

```
460 // Ustaw prędkość W<procent_0_100>
461 case 'w': // Komenda 'W' z parametrem procentu prędkości
462     if (command_char == 'w' && strlen(command_str) > 1) { // Sprawdzanie czy jest podany parametr
463         if (sscanf(command_str + 1, "%d", &value) == 1) {
464             if (value >= 0 && value <= 100) {
465                 // Obliczanie docelowego PWM dla lewego silnika (0-999) na podstawie procentu
466                 uint16_t target_pwm_left = (uint16_t)((value / 100.0f) * 999.0f);
467                 // Ograniczenie górne PWM do 999
468                 if (target_pwm_left > 999) target_pwm_left = 999;
469                 current_bt_pwm_left = target_pwm_left; // Ustawia nowe PWM dla lewego silnika
470                 // Obliczanie PWM dla prawego silnika używając stałej różnicy
471                 int16_t calculated_right_pwm = target_pwm_left - pwm_speed_diff;
472                 // Sprawdzanie czy PWM dla prawego silnika nie jest ujemne i nie przekracza 999
473                 if (calculated_right_pwm < 0) {
474                     current_bt_pwm_right = 0;
475                 } else if (calculated_right_pwm > 999) {
476                     current_bt_pwm_right = 999;
477                 } else {
478                     current_bt_pwm_right = (uint16_t)calculated_right_pwm; // Ustawia nowe PWM dla prawego silnika
479                 }
480                 snprintf(msg, sizeof(msg), "BT CMD: Predkosc ustawiona na %d%% (L:%d, P:%d)\r\n", value, current_bt_pwm_left, current_bt_pwm_right);
481                 UART_SendString(msg);
482             } else {
483                 UART_SendString("BT ERR: Procent predkosci poza zakresem (0-100)\r\n");
484             }
485         } else {
486             UART_SendString("BT ERR: Bledny format komendy predkosci. Uzyj W<0-100>\r\n");
487         }
488     } else if (command_char == 'w' && strlen(command_str) == 1) { // Samo 'w' bez parametru
489         UART_SendString("BT ERR: Komenda W wymaga procentu predkosci. Uzyj W<0-100>\r\n");
490     } else {
491         snprintf(msg, sizeof(msg), "BT CMD: Nieznana komenda '%s'\r\n", command_str);
492         UART_SendString(msg);
493     }
494     break;
495 default:
496     snprintf(msg, sizeof(msg), "BT CMD: Nieznana komenda '%s'\r\n", command_str);
497     UART_SendString(msg);
498     break;
499 }
```



robota

Oliwier Bogdański, Kacper Szponar

## Funkcja Main()

```
515 int main(void)
516 {
517     /* USER CODE BEGIN 1 */
518     /* USER CODE END 1 */
519
520     /* MCU Configuration-----*/
521
522     /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
523     HAL_Init();
524
525     /* USER CODE BEGIN Init */
526     /* USER CODE END Init */
527
528     /* Configure the system clock */
529     SystemClock_Config();
530
531     /* USER CODE BEGIN SysInit */
532     /* USER CODE END SysInit */
533
534     /* Initialize all configured peripherals */
535     MX_GPIO_Init();
536     MX_DMA_Init();
537     MX_TIM1_Init();
538     MX_TIM2_Init();
539     MX_USART1_UART_Init();
540     MX_ADC4_Init();
541     MX_USART2_UART_Init();
542     /* USER CODE BEGIN 2 */
543
544     // Inicjalizacja DFPlayer Mini
545     dfplayer_init(&huart2, GPIOC, GPIO_PIN_13);
546     HAL_Delay(500);
547
548     HAL_TIM_PWM_Start(&htim1, TIM_CHANNEL_1); // Start generowania sygnału PWM na kanale 1 (prawy silnik)
549     HAL_TIM_PWM_Start(&htim1, TIM_CHANNEL_2); // Start generowania sygnału PWM na kanale 2 (lewy silnik)
550
551     HAL_GPIO_WritePin(GPIOC, GPIO_PIN_0, GPIO_PIN_SET); // IrED = 1 -> Diody IR w czujniku odbiciowym włączone
552
553     line_follower_init(); // Inicjalizacja modułu śledzenia linii
554
555     //const uint16_t pwm_r = 980; // Minimalnie niższe wypełnienie PWM dla prawego silnika, by skorygować tor jazdy
556     //const uint16_t pwm_l = 999; // Maksymalne wypełnienie PWM (pełna prędkość) dla lewego silnika
557
558     // Uruchomienie nasłuchiwanie na porcie UART1 dla komend Bluetooth.
559     // Odbieranie pojedynczych bajtów do bufora bt_command_buffer.
560     HAL_UART_Receive_IT(&huart1, (uint8_t*)&bt_command_buffer[bt_cmd_buffer_idx], 1);
561
562     HAL_ADC_Start_DMA(&hadc4, (uint32_t*)adc_buffer, ADC_CHANNELS); // Uruchomienie konwersji ADC w trybie DMA
563     // Odczyty z `ADC_CHANNELS` kanałów będą
```





robota

Oliwier Bogdański, Kacper Szponar

```
551
552 HAL_TIM_PWM_Start(&htim1, TIM_CHANNEL_1); // Start generowania sygnału PWM na kanale 1 (prawy silnik)
553 HAL_TIM_PWM_Start(&htim1, TIM_CHANNEL_2); // Start generowania sygnału PWM na kanale 2 (lewy silnik)
554
555 HAL_GPIO_WritePin(GPIOC, GPIO_PIN_0, GPIO_PIN_SET); // IrED = 1 -> Diody IR w czujniku odbiciowym włączone
556
557 line_follower_init(); // Inicjalizacja modułu śledzenia linii
558
559 //const uint16_t pwm_r = 980; // Minimalnie niższe wypełnienie PWM dla prawego silnika, by skorygować tor jazdy
560 //const uint16_t pwm_l = 999; // Maksymalne wypełnienie PWM (pełna prędkość) dla lewego silnika
561
562 // Uruchomienie nasłuchiwanie na porcie UART1 dla komend Bluetooth.
563 // Odbieranie pojedynczych bajtów do bufora bt_command_buffer.
564 HAL_UART_Receive_IT(&huart1, (uint8_t*)&bt_command_buffer[bt_cmd_buffer_idx], 1);
565
566 HAL_ADC_Start_DMA(&hadc4, (uint32_t*)adc_buffer, ADC_CHANNELS); // Uruchomienie konwersji ADC w trybie DMA
567 // Odczyty z `ADC_CHANNELS` kanałów będą
568 // automatycznie zapisywane do `adc_buffer`
569
570 /* USER CODE END 2 */
571
572 /* Infinite loop */
573 /* USER CODE BEGIN WHILE */
574
575
576 // Ustawienie początkowego trybu i stanu
577 current_robot_mode = ROBOT_MODE_STOPPED; // Ustawienie początkowego trybu robota na zatrzymany
578 robot_running = 0; // Domyślnie zatrzymany
579
580 // Wysłanie komunikatów powitalnych przez UART
581 UART_SendString("Robot Gotowy!!!\r\nWyślij 'A' aby włączyć Podążanie po linii, 'M' dla ręcznego sterowania.\r\n");
582 UART_SendString("Aktualny Tryb: Zatrzymano. Wciśnij PA0 albo wyślij komendę BT.\r\n");
583
584 while (1)
585 { // sterowanie diodą state (PE8)
586     if (HC05_State()) {
587         HAL_GPIO_WritePin(GPIOE, GPIO_PIN_8, GPIO_PIN_SET); // Dioda ON (połączony)
588     }
589     else {
590         HAL_GPIO_WritePin(GPIOE, GPIO_PIN_8, GPIO_PIN_RESET); // Dioda OFF (brak połączenia)
591     }
592
593     /* Kontrola diody LED dla statusu BUSY DFPlayera */
594     if (dfplayer_is_busy()) {
595         // DFPlayer jest zajęty (gra muzykę) - zaświeć diodę na PE15
596         HAL_GPIO_WritePin(GPIOE, GPIO_PIN_15, GPIO_PIN_SET);
597     } else {
598         // DFPlayer jest wolny (nie gra) - zgaś diodę na PE15
599         HAL_GPIO_WritePin(GPIOE, GPIO_PIN_15, GPIO_PIN_RESET);
600     }
601 }
602 /* USER CODE END WHILE */
603
```



robota

Oliwier Bogdański, Kacper Szponar

```
347 if (HC05_State()) {
348     HAL_GPIO_WritePin(GPIOE, GPIO_PIN_8, GPIO_PIN_SET); // Dioda ON (połączony)
349 }
350 else {
351     HAL_GPIO_WritePin(GPIOE, GPIO_PIN_8, GPIO_PIN_RESET); // Dioda OFF (brak połączenia)
352 }
353 /* USER CODE END WHILE */
354
355 /* USER CODE BEGIN 3 */
356 // switch sterujący robotem w zależności od ustawionego trybu
357 switch (current_robot_mode) {
358     case ROBOT_MODE_LINE_FOLLOWER:
359         if (robot_running) {
360             float distance = HCSR04_ReadDistance(); // Pomiar dystansu
361
362             if (distance >= 16.0f || distance < 0.0f) { // Jeśli nie ma przeszkody lub błąd czujnika
363                 line_follower_process(); // Wywołanie funkcji śledzenia linii
364                 HAL_Delay(1); // krótkie opóźnienie
365             } else { // Przeszkoda wykryta
366                 robot_stop();
367                 UART_SendString("Przeszkoda: Zatrzymano.\r\n");
368                 // Opcjonalnie: Zawracanie
369                 // UART_SendString("Przeszkoda: Zawracam.\r\n");
370                 // HAL_Delay(500);
371                 // robot_drive(BT_PWM_SPEED_RIGHT, 1, BT_PWM_SPEED_LEFT, 0);
372                 // HAL_Delay(600);
373                 // robot_stop();
374                 // HAL_Delay(50);
375             }
376         } else {
377             robot_stop(); // zatrzymanie po wciśnięciu przycisku
378             line_follower_reset_state();
379             HAL_Delay(200);
380         }
381         break;
382
383     case ROBOT_MODE_BLUETOOTH_MANUAL:
384         // Logika ruchu jest obsługiwana w ProcessBluetoothCommand poprzez przerwanie UART
385         HAL_Delay(50); // Krótkie opóźnienie
386         break;
387
388     case ROBOT_MODE_STOPPED:
389         // Robot jest zatrzymany, silniki wyłączone
390         HAL_Delay(200); // odciążanie CPU
391         break;
392 }
393 /* USER CODE END 3 */
394 }
395 }
```



## robota

Oliwier Bogdański, Kacper Szponar

### Funkcje callback

```
8378 /* USER CODE BEGIN 4 */
838 // Funkcja obsługująca przerwanai zewnętrzne wywoływane przez wciśnięcie User Button (PA0)
839 void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)
840 {
841     if (GPIO_Pin == GPIO_PIN_0) // Sprawdza, czy przerwanie pochodzi od przycisku użytkownika (PA0)
842     {
843         // Prosty debounce
844         static uint32_t last_press_time = 0; // Zmienna przechowuje czas ostatniego zarejestrowanego naciśnięcia
845         if (HAL_GetTick() - last_press_time < 250) { // 250ms debounce
846             return; // Ignoruj to naciśnięcie
847         }
848         last_press_time = HAL_GetTick();
849
850         if (current_robot_mode == ROBOT_MODE_LINE_FOLLOWER) {
851             robot_running = !robot_running; // Przełącz stan start/stop dla line-followera
852             if (robot_running) {
853                 line_follower_reset_state(); // Resetowanie stanu przy starcie
854                 UART_SendString("PA0: START podążania po linii\r\n");
855             } else { // Zatrzymanie robota
856                 robot_stop();
857                 UART_SendString("PA0: STOP podążania po linii\r\n");
858             }
859         } else if (current_robot_mode == ROBOT_MODE_BLUETOOTH_MANUAL) {
860             // W trybie manualnym BT, PA0 działa jako nagły stop
861             robot_stop();
862             UART_SendString("PA0: Zatrzymano ręczne sterowanie\r\n");
863         } else if (current_robot_mode == ROBOT_MODE_STOPPED) {
864             // Jeśli zatrzymany, PA0 może go przełączyć i uruchomić w trybie Line Follower
865             current_robot_mode = ROBOT_MODE_LINE_FOLLOWER;
866             robot_running = 1; // Uruchomienie od razu
867             line_follower_reset_state();
868             UART_SendString("PA0: Zmieniono tryb na podążanie po linii\r\n");
869         }
870     }
871 }
872 }

1164 // funkcja obsługująca przerwanai od timera w trybie Input Capture (przechwytywania wejścia)
1165 // Wywoływana, gdy timer wykryje zbocze na skonfigurowanym kanale
1166 // służy do pomiaru czasu trwania impulsu echo z HC-SR04
1167 void HAL_TIM_IC_CaptureCallback(TIM_HandleTypeDef *htim)
1168 {
1169     // Sprawdzenie, czy przerwanie pochodzi od TIM2 i jego aktywnego kanału 4
1170     if (htim->Instance == TIM2 && htim->Channel == HAL_TIM_ACTIVE_CHANNEL_4)
1171     {
1172         if (echo_captured == 0) // Jeśli to pierwsze (narastające) zbocze impulsu echo
1173         {
1174             echo_start = HAL_TIM_ReadCapturedValue(htim, TIM_CHANNEL_4); // Odczytaj i zapisz wartość licznika timera (czas startu)
1175
1176             // Zmień polaryzację przechwytywania na zbocze opadające, aby złapać koniec impulsu
1177             __HAL_TIM_SET_CAPTUREPOLARITY(htim, TIM_CHANNEL_4, TIM_INPUTCHANNELPOLARITY_FALLING);
1178             echo_captured = 1; // Ustawia flagę, że start echa został przechwycony
1179         }
1180         else if (echo_captured == 1) // Jeśli to drugie (opadające) zbocze impulsu echo
1181         {
1182             echo_end = HAL_TIM_ReadCapturedValue(htim, TIM_CHANNEL_4); // Odczytaj i zapisz wartość licznika timera (czas końca)
1183
1184             // Zmień polaryzację z powrotem na zbocze narastające dla następnego pomiaru
1185             __HAL_TIM_SET_CAPTUREPOLARITY(htim, TIM_CHANNEL_4, TIM_INPUTCHANNELPOLARITY_RISING);
1186             echo_captured = 2; // Ustaw flagę, że koniec echa został przechwycony (pomiar kompletny)
1187         }
1188     }
1189 }
1190 }
1191 }
```



robota

Oliwier Bogdański, Kacper Szponar

```
1193 // Callback wywołany po odebraniu danych przez UART (Bluetooth)
1194 void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart)
1195 {
1196     if (huart->Instance == USART1) // Sprawdź, czy przerwanie pochodzi od USART1
1197     {
1198         uint8_t received_char = bt_command_buffer[bt_cmd_buffer_idx]; // Pobierz właśnie odebrany znak
1199
1200         // Sprawdzenie czy odebrano znak końca linii
1201         if (received_char == '\n') // Jeśli odebrano LF
1202         {
1203             bt_command_buffer[bt_cmd_buffer_idx] = '\0'; // Kończenie stringa w miejscu LF
1204             // Uniknięcie pustych komend poprzez zamianę CR na \0
1205             if (bt_cmd_buffer_idx > 0 && bt_command_buffer[bt_cmd_buffer_idx - 1] == '\r') {
1206                 bt_command_buffer[bt_cmd_buffer_idx - 1] = '\0';
1207             }
1208             if (strlen(bt_command_buffer) > 0) { // Przetwarza tylko jeśli komenda nie jest pusta
1209                 ProcessBluetoothCommand(bt_command_buffer);
1210             }
1211             bt_cmd_buffer_idx = 0; // Reset indeksu dla nowej komendy
1212             memset(bt_command_buffer, 0, BT_COMMAND_BUFFER_SIZE); // Czyszczenie buforu
1213         }
1214         else if (received_char == '\r') // Jeśli odebrano CR
1215         {
1216             bt_command_buffer[bt_cmd_buffer_idx] = '\0'; // Kończenie stringa w miejscu CR
1217             // Komenda jest gotowa do przetworzenia
1218             if (strlen(bt_command_buffer) > 0) {
1219                 ProcessBluetoothCommand(bt_command_buffer);
1220             }
1221             bt_cmd_buffer_idx = 0; // Reset indeksu dla nowej komendy
1222             memset(bt_command_buffer, 0, BT_COMMAND_BUFFER_SIZE); // Czyszczenie bufora
1223         }
1224         else
1225         {
1226             // Jeśli to zwykły znak to dodaje do bufora i inkrementuje indeks
1227             // Sprawdzenie, czy nie przekroczono rozmiaru bufora
1228             if (bt_cmd_buffer_idx < BT_COMMAND_BUFFER_SIZE - 1) {
1229                 bt_cmd_buffer_idx++;
1230             } else {
1231                 // Jeśli bufor jest pełny a nie było znaku końca to błąd
1232                 UART_SendString("BT ERR: Bufor komendy przepełniony\r\n");
1233                 bt_cmd_buffer_idx = 0; // Reset buforu
1234                 memset(bt_command_buffer, 0, BT_COMMAND_BUFFER_SIZE);
1235             }
1236         }
1237     }
1238     // Ponowne włączenie nasłuchiwanie i zapisanie go w aktualnej pozycji w buforze
1239     HAL_UART_Receive_IT(&huart1, (uint8_t*)&bt_command_buffer[bt_cmd_buffer_idx], 1);
1240 }
1241 }
1242 }
```



## robotą

Oliwier Bogdański, Kacper Szponar

### Line\_follower.h

plik nagłówkowy dla line\_follower zawierający definicje i parametry

```
1 /*  
2  * line_follower.h  
3  */  
4  
5 #ifndef LINE_FOLLOWER_H_  
6 #define LINE_FOLLOWER_H_  
7  
8 #include "main.h"  
9 #include <stdint.h>  
10  
11 //Konfiguracja śledzenia linii  
12 #define ADC_CHANNELS 8 // Liczba czujników linii (kanałów ADC)  
13  
14 #define LINE_FOLLOWER_KP 0.001f // Współczynnik proporcjonalny (P) regulatora PID do korekcji błędu pozycji  
15 #define LINE_FOLLOWER_KD 0.001f // Współczynnik różniczkujący (D) regulatora PID do tłumienia oscylacji  
16  
17 #define LINE_FOLLOWER_BASE_SPEED_PWM 650 // Podstawowa prędkość PWM silników  
18 #define LINE_FOLLOWER_MIN_SPEED_PWM 650 // Minimalna dopuszczalna prędkość PWM silników  
19 #define LINE_FOLLOWER_MAX_SPEED_PWM 850 // Maksymalna dopuszczalna prędkość PWM silników  
20  
21 // Definicja bazowych prędkości PWM dla obu silników uwzględniając kalibrację  
22 #define LEFT_MOTOR_BASE_PWM (LINE_FOLLOWER_BASE_SPEED_PWM + 20 > LINE_FOLLOWER_MAX_SPEED_PWM ? LINE_FOLLOWER_MAX_SPEED_PWM : LINE_FOLLOWER_BASE_SPEED_PWM + 20)  
23 #define RIGHT_MOTOR_BASE_PWM LINE_FOLLOWER_BASE_SPEED_PWM  
24  
25 #define SENSOR_LINE_THRESHOLD 600 // Próg odczytu ADC, powyżej którego uznaje się,  
26 // że czujnik jest nad linią (białe tło, czarna linia)  
27 #define SENSOR_WEIGHT_MULTIPLIER 1000 // Mnożnik używany do obliczania ważonej pozycji linii  
28  
29 // Docelowa wartość pozycji, gdy robot jest idealnie na środku linii  
30 #define LINE_CENTER_POSITION ( ( ADC_CHANNELS - 1 ) * SENSOR_WEIGHT_MULTIPLIER ) / 2 )  
31  
32 //Konfiguracja ostrych zakrętów  
33 #define SHARP_TURN_PWM 700 // Prędkość PWM silników podczas wykonywania ostrego zakrętu  
34 #define SHARP_TURN_DURATION_MS 600 // Czas trwania manewru ostrego zakrętu w milisekundach  
35 #define SHARP_TURN_OUTER_SENSORS_ACTIVE 2 // Liczba skrajnych zewnętrznych czujników,  
36 // które muszą wykryć linię, aby zidentyfikować ostry zakręt  
37  
38 #define SHARP_TURN_INNER_SENSORS_INACTIVE 3 // Liczba wewnętrznych czujników (od strony zakrętu),  
39 // które muszą NIE wykrywać linii,  
40 // aby potwierdzić ostry zakręt  
41  
42 //Konfiguracja oszukiwania zgubionej linii  
43 #define LOST_LINE_REVERSE_PWM 710 // Prędkość PWM podczas cofania  
44 #define LOST_LINE_REVERSE_DURATION_MS 400 // Czas cofania po zgubieniu linii  
45 #define LOST_LINE_SWEEP_TURN_PWM 550 // Prędkość PWM podczas obrotu w miejscu przy szukaniu  
46 #define LOST_LINE_SWEEP_DURATION_MS 550 // Czas jednego obrotu przy szukaniu  
47 #define LOST_LINE_MAX_SEARCH_CYCLES 5 // Liczba pełnych cykli poszukiwania (lewo-prawo) zanim robot się podda  
48  
49  
50 // Stany robota  
51 typedef enum {  
52     STATE_FOLLOWING_LINE, // Stan gdy robot jedzie po linii  
53  
54     // Stany dla zgubionej linii  
55     STATE_LOST_LINE_INITIATE_SEARCH, // Stan początkowy po zgubieniu linii  
56     STATE_LOST_LINE_REVERSING, // Robot cofa  
57     STATE_LOST_LINE_SEARCH_LEFT, // Robot szuka obracając się w lewo w miejscu  
58     STATE_LOST_LINE_SEARCH_RIGHT, // Robot szuka obracając się w prawo w miejscu  
59     STATE_LOST_LINE_FAIL_STOP, // Robot się poddaje i zatrzymuje  
60  
61     // Stany dla ostrych zakrętów  
62     STATE_DETECTED_SHARP_LEFT_TURN,  
63     STATE_PERFORMING_SHARP_LEFT_TURN,  
64     STATE_DETECTED_SHARP_RIGHT_TURN,  
65     STATE_PERFORMING_SHARP_RIGHT_TURN,  
66 } RobotLineState;  
67  
68 extern uint16_t adc_buffer[ADC_CHANNELS]; // Bufor przechowujący odczyty z czujników linii (ADC)  
69  
70 void line_follower_init(void); // Funkcja inicjalizująca moduł śledzenia linii  
71 void line_follower_process(void); // Główna funkcja przetwarzająca logikę śledzenia linii, wywoływana cyklicznie  
72 void line_follower_reset_state(void); // Funkcja resetująca stan robota do początkowego (np. STATE_FOLLOWING_LINE)  
73  
74 #endif /* LINE_FOLLOWER_H_ */  
75
```



robota

Oliwier Bogdański, Kacper Szponar

## Line\_follower.c

### Zmienne

```
1 #include "line_follower.h" //import zmiennych i definicji z pliku nagłówkowego
2 #include "main.h"
3 #include <stdio.h>
4 #include <string.h>
5 #include <stdlib.h>
6
7 // Tablica wag przypisanych do każdego czujnika. Wagi rosną od lewej do prawej,
8 // co pozwala na obliczenie środka ciężkości wykrytej linii.
9 static const int32_t sensor_weights[ADC_CHANNELS] = {
10     0 * SENSOR_WEIGHT_MULTIPLIER, 1 * SENSOR_WEIGHT_MULTIPLIER, 2 * SENSOR_WEIGHT_MULTIPLIER, 3 * SENSOR_WEIGHT_MULTIPLIER,
11     4 * SENSOR_WEIGHT_MULTIPLIER, 5 * SENSOR_WEIGHT_MULTIPLIER, 6 * SENSOR_WEIGHT_MULTIPLIER, 7 * SENSOR_WEIGHT_MULTIPLIER
12 };
13
14 // Statyczne zmienne modułu przechowujące jego wewnętrzny stan
15
16 static float line_last_error_for_decision = 0.0f; // Ostatni zarejestrowany błąd pozycji linii,
17 // używany do podjęcia decyzji o kierunku poszukiwania linii po jej zgubieniu.
18 // Aktualizowany tylko, gdy linia jest wykrywana stabilnie
19
20 static float line_last_error_for_pid = 0.0f; // Ostatni zarejestrowany błąd pozycji linii,
21 // używany do obliczenia członu różniczkującego regulatora PID
22
23 static int32_t last_known_position = LINE_CENTER_POSITION; // Ostatnia znana pozycja linii, nawet jeśli była chwilowa
24 static RobotLineState_t current_robot_state = STATE_FOLLOWING_LINE; // Aktualny stan robota
25 static uint32_t state_timer = 0; // Zmienna używana do odmierzania czasu trwania określonych stanów lub operacji
26 static uint8_t search_cycle_counter = 0; // Licznik cykli poszukiwania wykonanych podczas szukania zgubionej linii
27
```

### Funkcje pomocnicze

```
28 // Prototypy funkcji prywatnych
29 static int32_t calculate_line_position(uint16_t *sensor_values);
30 static void handle_lost_line(void);
31 static void handle_sharp_turns(int32_t position, uint16_t *sensor_values);
32
33
34 // Funkcja pomocnicza inicjalizująca stan modułu szukania linii. Wywołana tylko raz
35
36 void line_follower_init(void) {
37     line_last_error_for_decision = 0.0f;
38     line_last_error_for_pid = 0.0f;
39     last_known_position = LINE_CENTER_POSITION; // Zakładamy, że na starcie robot jest na środku linii
40     current_robot_state = STATE_FOLLOWING_LINE; // Domyślny stan początkowy.
41     state_timer = HAL_GetTick(); // Inicjalizacja prostego timera stanów za pomocą HAL
42     search_cycle_counter = 0; // Zmienna zliczająca ilość cykli poszukiwania
43 }
44
45 // Resetuje stan modułu, pozwalając na uruchomienie od nowa podążania za linią
46 void line_follower_reset_state(void) {
47     line_last_error_for_decision = 0.0f;
48     line_last_error_for_pid = 0.0f;
49     current_robot_state = STATE_FOLLOWING_LINE;
50     search_cycle_counter = 0;
51 }
52
```



robota

Oliwier Bogdański, Kacper Szponar

```
52
53 // Funkcja pomocnicza obliczająca pozycję linii na podstawie odczytów z czujników odbiciowych
54 // Zwraca wartość pozycji lub -1, jeśli linia nie została wykryta.
55
56 static int32_t calculate_line_position(uint16_t *sensor_values) {
57     uint32_t weighted_sum = 0; // Suma ważona (wartość czujnika * waga czujnika)
58     uint16_t sum_of_active_sensors = 0; // Suma wartości aktywnych czujników
59     uint8_t active_sensor_count = 0; // Liczba czujników, które wykryły linię
60     uint8_t first_active_sensor = ADC_CHANNELS; // Indeks pierwszego aktywnego czujnika od lewej
61
62     // Czujnik jest uznawany za aktywny, jeśli jego odczyt przekracza próg SENSOR_LINE_THRESHOLD
63     for (uint8_t i = 0; i < ADC_CHANNELS; i++) {
64         if (sensor_values[i] > SENSOR_LINE_THRESHOLD) {
65             weighted_sum += (uint32_t)sensor_values[i] * sensor_weights[i];
66             sum_of_active_sensors += sensor_values[i];
67             active_sensor_count++;
68             if (i < first_active_sensor) { // Zapamiętanie indeksu pierwszego aktywnego czujnika.
69                 first_active_sensor = i;
70             }
71         }
72     }
73
74     if (active_sensor_count == 0) {
75         return -1; // -1 oznacza że, linia nie została wykryta przez żaden czujnik
76     }
77
78     // Jeśli wszystkie czujniki widzą linię (np. szeroka linia) to uznajemy, że robot jest na środku
79     if (active_sensor_count == ADC_CHANNELS) {
80         last_known_position = LINE_CENTER_POSITION;
81         return LINE_CENTER_POSITION;
82     }
83
84     // Jeśli tylko jeden czujnik jest aktywny, jego pozycja jest traktowana jako pozycja linii.
85     if (active_sensor_count == 1) {
86         if (first_active_sensor == 0) { // Aktywny tylko skrajny lewy czujnik.
87             last_known_position = sensor_weights[0];
88             return sensor_weights[0];
89         }
90         if (first_active_sensor == ADC_CHANNELS - 1) { // Aktywny tylko skrajny prawy czujnik.
91             last_known_position = sensor_weights[ADC_CHANNELS - 1];
92             return sensor_weights[ADC_CHANNELS - 1];
93         }
94     }
95
96     // Jeśli aktywny jest jeden ze środkowych czujników to obliczana jest średnia ważona
97     // Normalizacja wyniku daje większą wagę czujnikom z silniejszym sygnałem
98     last_known_position = weighted_sum / sum_of_active_sensors;
99     return last_known_position; // Zwracamy pozycję
100 }
```





robota

Oliwier Bogdański, Kacper Szponar

## Główna logika

```
101 // Główna funkcja przetwarzająca logikę podążania za linią wywoływana cyklicznie
102
103 void line_follower_process(void) {
104     // Obliczanie aktualnej pozycji linii na podstawie danych z bufora ADC
105     int32_t current_position = calculate_line_position(adc_buffer);
106
107     // Debug UART do sprawdzania wartości wskazywanych przez czujniki
108     char uart_buf[120];
109     sprintf(uart_buf, "ADC:[%4u %4u %4u %4u %4u %4u %4u %4u] Pos:%ld State:%d\r\n",
110         adc_buffer[0], adc_buffer[1], adc_buffer[2], adc_buffer[3],
111         adc_buffer[4], adc_buffer[5], adc_buffer[6], adc_buffer[7],
112         current_position, current_robot_state);
113     UART_SendString(uart_buf);
114
115     // W pierwszej kolejności sprawdzamy czy nie ma zakrętu
116     handle_sharp_turns(current_position, adc_buffer);
117     // Jeśli robot jest w trakcie zakrętu pomijamy resztę logiki
118     if (current_robot_state == STATE_PERFORMING_SHARP_LEFT_TURN ||
119         current_robot_state == STATE_PERFORMING_SHARP_RIGHT_TURN) {
120         return;
121     }
122
123     // Linia jest znaleziona
124     if (current_position != -1) {
125
126         current_robot_state = STATE_FOLLOWING_LINE; // Ustawienie stanu robota na podążanie za linią
127         search_cycle_counter = 0; // Zresetowanie licznika cykli poszukiwania bo linia została znaleziona
128
129         // Obliczenie błędu jako odchylenia aktualnej pozycji robota od idealnej pozycji linii
130         float error = (float)current_position - LINE_CENTER_POSITION;
131
132         // Aktualizacja zmiennej line_last_error_for_decision tylko wtedy,
133         // gdy błąd nie jest ekstremalny czyli np kiedy linia nie jest na skraju
134         uint8_t active_sensors_count_temp = 0;
135         for(int i=0; i<ADC_CHANNELS; ++i) if(adc_buffer[i] > SENSOR_LINE_THRESHOLD) active_sensors_count_temp++;
136
137         if (abs((int)error) < (LINE_CENTER_POSITION * 0.85f) && active_sensors_count_temp > 0) {
138             line_last_error_for_decision = error;
139         }
140
141         // Obliczenie członu różniczkującego (D) regulatora PID.
142         float derivative = error - line_last_error_for_pid;
143         // Obliczenie korekty prędkości silników za pomocą regulatora PD (Proporcjonalno-Różniczkującego)
144         float motor_speed_correction = (LINE_FOLLOWER_KP * error) + (LINE_FOLLOWER_KD * derivative);
145         // Zapisanie bieżącego błędu do użycia w następnej iteracji jako line_last_error_for_pid
146         line_last_error_for_pid = error;
```



robota

Oliwier Bogdański, Kacper Szponar

```
147
148 // Obliczenie docelowych wartości PWM dla silników
149
150 // Korekta jest odejmowana od lewego silnika i dodawana do prawego i
151 // powoduje to skręt w kierunku przeciwnym do znaku błędu
152 int16_t pwm_left_target = LEFT_MOTOR_BASE_PWM - (int16_t)motor_speed_correction;
153 int16_t pwm_right_target = RIGHT_MOTOR_BASE_PWM + (int16_t)motor_speed_correction;
154
155 // Ograniczenie prędkości silników do zdefiniowanych limitów w line_follower.h
156 if (pwm_left_target > LINE_FOLLOWER_MAX_SPEED_PWM) pwm_left_target = LINE_FOLLOWER_MAX_SPEED_PWM;
157 else if (pwm_left_target < LINE_FOLLOWER_MIN_SPEED_PWM) pwm_left_target = LINE_FOLLOWER_MIN_SPEED_PWM;
158
159 if (pwm_right_target > LINE_FOLLOWER_MAX_SPEED_PWM) pwm_right_target = LINE_FOLLOWER_MAX_SPEED_PWM;
160 else if (pwm_right_target < LINE_FOLLOWER_MIN_SPEED_PWM) pwm_right_target = LINE_FOLLOWER_MIN_SPEED_PWM;
161
162 // Dodatkowa korekta dla bardzo dużych błędów (min sytuacje gdy linia jest na skrajnych czujnikach)
163 // Ma to na celu wymuszenie ostrzejszego skrętu
164 // zmienna 'extreme_error_threshold' określa próg dla bardzo dużego błędu
165
166 const int32_t extreme_error_threshold = (ADC_CHANNELS / 2 - 1) * SENSOR_WEIGHT_MULTIPLIER * 0.8f;
167 if (abs((int)error) > extreme_error_threshold) {
168     if (error > 0) { // Linia mocno po prawej, robot musi skręcić w prawo (lewe koło szybciej, prawe wolniej)
169         pwm_left_target = LINE_FOLLOWER_MAX_SPEED_PWM;
170         pwm_right_target = LINE_FOLLOWER_MIN_SPEED_PWM - 200 > 0 ? LINE_FOLLOWER_MIN_SPEED_PWM - 200 : 0; // Zmniejszenie prędkości prawego
171     } else { // Linia mocno po lewej, robot musi skręcić w lewo
172         pwm_right_target = LINE_FOLLOWER_MAX_SPEED_PWM;
173         pwm_left_target = LINE_FOLLOWER_MIN_SPEED_PWM - 200 > 0 ? LINE_FOLLOWER_MIN_SPEED_PWM - 200 : 0;
174     }
175 }
176
177 // Sterowanie silnikami z obliczonymi wartościami PWM i kierunkiem
178 robot_drive(pwm_right_target, 1, pwm_left_target, 1);
179
180 // Na końcu sprawdzamy czy linia nie została zgubiona (current_position == -1)
181 } else {
182     // Jeśli robot nie jest już w stanie poszukiwania lub wykonywania ostrego zakrętu,
183     // a był wcześniej w stanie podążania za linią lub właśnie wykrył zakręt i zgubił linię,
184     // to inicjujemy procedurę poszukiwania
185     if (current_robot_state == STATE_FOLLOWING_LINE || current_robot_state == STATE_DETECTED_SHARP_LEFT_TURN ||
186         current_robot_state == STATE_DETECTED_SHARP_RIGHT_TURN) {
187
188         current_robot_state = STATE_LOST_LINE_INITIATE_SEARCH;
189
190         // Zmienne 'line_last_error_for_decision' i 'last_known_position'
191         // przechowują wartości sprzed zgubienia linii i zostaną użyte w funkcji handle_lost_line
192     }
193     handle_lost_line(); // Jeśli linia zgubiona to wywołujemy funkcję obsługującą ten stan
194 }
195 }
196
```



robota

Oliwier Bogdański, Kacper Szponar

Funkcja obsługująca zgubienie linii

```
197 // Funkcja obsługująca logikę zachowania robota po zgubieniu linii
198 // Linia jest poszukiwana sekwencyjnie do momentu aż robot znajdzie linię w ciągu ustalonej liczby cykli poszukiwania
199 static void handle_lost_line(void) {
200     uint32_t current_time = HAL_GetTick(); // Pobranie aktualnego czasu systemowego.
201
202     switch (current_robot_state) {
203         // Inicjalizacja sekwencji
204         case STATE_LOST_LINE_INITIATE_SEARCH:
205             // Najpierw zaczyna on cofnięcia
206             current_robot_state = STATE_LOST_LINE_REVERSING;
207             state_timer = current_time; // Zapisz czas rozpoczęcia cofania.
208             search_cycle_counter = 0; // Zresetuj licznik cykli dla nowej sekwencji poszukiwania
209             robot_drive(LOST_LINE_REVERSE_PWM, 0, LOST_LINE_REVERSE_PWM, 0); // Cofanie prosto z ustalonym PWM dla cofania
210             break;
211
212         case STATE_LOST_LINE_REVERSING:
213             // Sprawdzanie czy nie upłynął czas przeznaczony na cofanie
214             if (current_time - state_timer > LOST_LINE_REVERSE_DURATION_MS) {
215                 // Po cofnięciu robot decyduje o pierwszym kierunku obrotu w miejscu
216                 // Decyzja jest podjęta na podstawie wartości `line_last_error_for_decision` lub, jeśli jest on bliski zera, na `last_known_position`
217                 // Jeśli linia była po prawej (błąd > 0 lub LKP > centrum), to zaczyna szukać w prawo
218                 // LKP - last_known_position
219
220                 if (line_last_error_for_decision > SENSOR_WEIGHT_MULTIPLIER / 4 || (abs((int)line_last_error_for_decision) <=
221                     SENSOR_WEIGHT_MULTIPLIER / 4 && last_known_position > LINE_CENTER_POSITION)) {
222                     current_robot_state = STATE_LOST_LINE_SEARCH_RIGHT;
223                     robot_drive(LOST_LINE_SWEEP_TURN_PWM, 0, LOST_LINE_SWEEP_TURN_PWM, 1); // Obrót w prawo
224                 } else { // Linia była po lewej, na środku, lub błąd był nieznaczący i LKP <= centrum
225                     current_robot_state = STATE_LOST_LINE_SEARCH_LEFT;
226                     robot_drive(LOST_LINE_SWEEP_TURN_PWM, 1, LOST_LINE_SWEEP_TURN_PWM, 0); // Obrót w lewo
227                 }
228                 state_timer = current_time; // Zapisz czas rozpoczęcia obrotu
229             } else {
230                 // Kontynuuj cofanie, jeśli czas nie upłynął
231                 robot_drive(LOST_LINE_REVERSE_PWM, 0, LOST_LINE_REVERSE_PWM, 0);
232             }
233             break;
234     }
235 }
```



robota

Oliwier Bogdański, Kacper Szponar

```
235
236     case STATE_LOST_LINE_SEARCH_LEFT:
237         // Sprawdzanie czy upłynął czas przeznaczony na obrót w lewo
238         if (current_time - state_timer > LOST_LINE_SWEEP_DURATION_MS) {
239             // Zakończono obrót w lewo robot przełącza się na obrót w prawo
240             current_robot_state = STATE_LOST_LINE_SEARCH_RIGHT;
241             state_timer = current_time;
242             robot_drive(LOST_LINE_SWEEP_TURN_PWM, 0, LOST_LINE_SWEEP_TURN_PWM, 1); // Obrót w prawo
243             UART_SendString("Lost: Sweeping Right (from Left)\r\n");
244             search_cycle_counter++; // Inkrementacja licznika cykli
245         } else {
246             // Kontynuacja obrotu w lewo
247             robot_drive(LOST_LINE_SWEEP_TURN_PWM, 1, LOST_LINE_SWEEP_TURN_PWM, 0);
248         }
249         break;
250
251     case STATE_LOST_LINE_SEARCH_RIGHT:
252         // Sprawdzanie czy upłynął czas przeznaczony na obrót w prawo
253         if (current_time - state_timer > LOST_LINE_SWEEP_DURATION_MS) {
254             // Zakończono obrót w prawo
255             // Sprawdzanie czy nie przekroczono maksymalnej liczby cykli poszukiwania
256             // mnożnik *2 bo jeden cykl to obrót w lewo i w prawo, -1 bo licznik zaczyna od 0
257             if (search_cycle_counter >= LOST_LINE_MAX_SEARCH_CYCLES * 2 - 1) {
258                 current_robot_state = STATE_LOST_LINE_FAIL_STOP;
259                 robot_stop(); // Zatrzymaj robota.
260                 UART_SendString("Linia zgubiona: Koniec cykli poszukiwania. Zatrzymano Robota.\r\n");
261             } else {
262                 // Robot przełącza się na obrót w lewo, kontynuując poszukiwanie.
263                 current_robot_state = STATE_LOST_LINE_SEARCH_LEFT;
264                 state_timer = current_time;
265                 robot_drive(LOST_LINE_SWEEP_TURN_PWM, 1, LOST_LINE_SWEEP_TURN_PWM, 0); // Obrót w lewo
266                 search_cycle_counter++; // Inkrementuj licznik cykli (pół cyklu)
267             }
268         } else {
269             // Kontynuacja obrotu w prawo
270             robot_drive(LOST_LINE_SWEEP_TURN_PWM, 0, LOST_LINE_SWEEP_TURN_PWM, 1);
271         }
272         break;
273
274     case STATE_LOST_LINE_FAIL_STOP:
275         robot_stop(); // Zatrzymanie robota
276         break;
277
278     default:
279         // W przypadku nieoczekiwanego stanu inicjujemy poszukiwanie jeszcze raz
280         current_robot_state = STATE_LOST_LINE_INITIATE_SEARCH;
281         break;
282 }
283 }
```



robotą

Oliwier Bogdański, Kacper Szponar

## Zakręty

```
285 // Funkcja obsługująca wykrywanie i wykonywanie ostrych zakrętów.
286 static void handle_sharp_turns(int32_t position, uint16_t *sensor_values) {
287     uint8_t active_left_outer = 0; // Liczba aktywnych czujników na skrajnej lewej stronie
288     uint8_t active_right_outer = 0; // Liczba aktywnych czujników na skrajnej prawej stronie
289     uint8_t active_center = 0; // Liczba aktywnych czujników w środkowej części
290     uint8_t total_active = 0; // Całkowita liczba aktywnych czujników
291
292     // Zliczanie aktywnych czujników w poszczególnych strefach
293     // Zmienna SHARP_TURN_OUTER_SENSORS_ACTIVE definiuje, ile skrajnych czujników jest branych pod uwagę
294     for(int i = 0; i < ADC_CHANNELS; i++) {
295         if (sensor_values[i] > SENSOR_LINE_THRESHOLD) {
296             total_active++;
297             if (i < SHARP_TURN_OUTER_SENSORS_ACTIVE) active_left_outer++;
298             else if (i >= ADC_CHANNELS - SHARP_TURN_OUTER_SENSORS_ACTIVE) active_right_outer++;
299             else active_center++;
300         }
301     }
302
303     uint32_t current_time = HAL_GetTick(); // Pobranie czasu
304
305     // Logika wykonywania ostrego zakrętu w lewo
306     if (current_robot_state == STATE_PERFORMING_SHARP_LEFT_TURN) {
307         // Zakończenie manewru skretu, jeśli upłynął zdefiniowany czas lub jeśli środkowe czujniki ponownie wykryły linię
308         if (current_time - state_timer > SHARP_TURN_DURATION_MS || (active_center > 0 && total_active < ADC_CHANNELS - 1 && total_active > 0)) {
309             int32_t pos_after_turn = calculate_line_position(sensor_values); // Sprawdzamy pozycję linii po zakręcie
310             // Jeśli linia jest znaleziona i widoczna przez środkowe czujniki to ustawiamy odpowiedni stan
311             if (pos_after_turn != -1 && active_center > 0) {
312                 current_robot_state = STATE_FOLLOWING_LINE;
313                 line_last_error_for_pid = 0; // Reset błędu PID w celu uniknięcia gwałtownej reakcji
314                 robot_drive(LINE_FOLLOWER_BASE_SPEED_PWM, 1, LINE_FOLLOWER_BASE_SPEED_PWM, 1);
315             } else { // Linia nie została znaleziona lub nie jest widoczna centralnie po zakręcie
316                 current_robot_state = STATE_LOST_LINE_INITIATE_SEARCH; // Rozpocznij poszukiwanie linii
317                 // handle_lost_line zostanie wywołane w głównej pętli line_follower_process
318             }
319         } else {
320             // Kontynuacja wykonywania ostrego skretu w lewo
321             robot_drive(SHARP_TURN_PWM, 1, (uint16_t)(SHARP_TURN_PWM * 0.3f), 0);
322         }
323         return; // Zakończenie jeśli wykonujemy zakręt
324     }
325
326     // Logika wykonywania ostrego zakrętu w prawo podobnie jak w lewo
327     if (current_robot_state == STATE_PERFORMING_SHARP_RIGHT_TURN) {
328         if (current_time - state_timer > SHARP_TURN_DURATION_MS || (active_center > 0 && total_active < ADC_CHANNELS - 1 && total_active > 0)) {
329             int32_t pos_after_turn = calculate_line_position(sensor_values);
330             if (pos_after_turn != -1 && active_center > 0) {
331                 current_robot_state = STATE_FOLLOWING_LINE;
332                 line_last_error_for_pid = 0;
333                 robot_drive(LINE_FOLLOWER_BASE_SPEED_PWM, 1, LINE_FOLLOWER_BASE_SPEED_PWM, 1);
334             } else {
335                 current_robot_state = STATE_LOST_LINE_INITIATE_SEARCH;
```



robota

Oliwier Bogdański, Kacper Szponar

```
331         current_robot_state = STATE_FOLLOWING_LINE;
332         line_last_error_for_pid = 0;
333         robot_drive(LINE_FOLLOWER_BASE_SPEED_PWM, 1, LINE_FOLLOWER_BASE_SPEED_PWM, 1);
334     } else {
335         current_robot_state = STATE_LOST_LINE_INITIATE_SEARCH;
336     }
337 } else {
338     robot_drive((uint16_t)(SHARP_TURN_PWM * 0.3f), 0, SHARP_TURN_PWM, 1);
339 }
340 return;
341 }
342
343 // Logika wykrywania ostrych zakrętów (tylko jeśli robot jest w stanie STATE_FOLLOWING_LINE i widzi linię)
344 if (current_robot_state == STATE_FOLLOWING_LINE && position != -1) {
345     // Warunek dla ostrego zakrętu w lewo:
346     // - Co najmniej SHARP_TURN_OUTER_SENSORS_ACTIVE skrajnych lewych czujników jest aktywnych
347     // - Maksymalnie 1 aktywny czujnik w centrum (aby odróżnić od prostego odcinka)
348     // - Całkowita liczba aktywnych czujników mieści się w pewnym zakresie
349     // - Obliczona pozycja linii jest zdecydowanie po lewej stronie (mniejsza niż 1/3 LINE_CENTER_POSITION).
350     if (active_left_outer >= SHARP_TURN_OUTER_SENSORS_ACTIVE &&
351         active_center <= 1 &&
352         total_active >= SHARP_TURN_OUTER_SENSORS_ACTIVE && total_active <= SHARP_TURN_OUTER_SENSORS_ACTIVE + 2 &&
353         position < (LINE_CENTER_POSITION / 3) )
354     {
355         current_robot_state = STATE_DETECTED_SHARP_LEFT_TURN;
356     }
357     // Warunek dla ostrego zakrętu w prawo (analogiczny):
358     // - Obliczona pozycja linii jest zdecydowanie po prawej stronie (większa niż centrum + 2/3 odległości od centrum do prawej krawędzi)
359     else if (active_right_outer >= SHARP_TURN_OUTER_SENSORS_ACTIVE &&
360             active_center <= 1 &&
361             total_active >= SHARP_TURN_OUTER_SENSORS_ACTIVE && total_active <= SHARP_TURN_OUTER_SENSORS_ACTIVE + 2 &&
362             position > (LINE_CENTER_POSITION + (LINE_CENTER_POSITION / 3) * 2) )
363     {
364         current_robot_state = STATE_DETECTED_SHARP_RIGHT_TURN;
365     }
366 }
367
368 // Rozpoczęcie wykonywania ostrego zakrętu po jego wykryciu
369 if (current_robot_state == STATE_DETECTED_SHARP_LEFT_TURN) {
370     current_robot_state = STATE_PERFORMING_SHARP_LEFT_TURN; // Zmiana stanu na wykonywanie.
371     state_timer = HAL_GetTick(); // Zapisanie czasu rozpoczęcia manewru
372     robot_drive(SHARP_TURN_PWM, 1, (uint16_t)(SHARP_TURN_PWM * 0.3f), 0);
373     return;
374 }
375
376 if (current_robot_state == STATE_DETECTED_SHARP_RIGHT_TURN) {
377     current_robot_state = STATE_PERFORMING_SHARP_RIGHT_TURN;
378     state_timer = HAL_GetTick();
379     robot_drive((uint16_t)(SHARP_TURN_PWM * 0.3f), 0, SHARP_TURN_PWM, 1);
380     return;
381 }
382 }
383 }
```



roboty

Oliwier Bogdański, Kacper Szponar

## Dfplayer\_mini.h

```
1  /*
2   * dfplayer_mini.h
3   */
4
5  #ifndef INC_DFPLAYER_MINI_H_
6  #define INC_DFPLAYER_MINI_H_
7
8  #include "stm32f3xx_hal.h"
9
10 // Komendy DFPlayerMini
11 #define DFPLAYER_CMD_PLAY_TRACK 0x03 // Odtwórz konkretny utwór (folder/plik)
12 #define DFPLAYER_CMD_SET_VOLUME 0x06 // Ustaw głośność (0-30)
13 #define DFPLAYER_CMD_SET_EQ 0x07 // Ustaw EQ (0:Normal, 1:Pop, 2:Rock, 3:Jazz, 4:Classic, 5:Base)
14 #define DFPLAYER_CMD_NEXT_TRACK 0x01 // Następny utwór
15 #define DFPLAYER_CMD_PREV_TRACK 0x02 // Poprzedni utwór
16 #define DFPLAYER_CMD_PLAY 0x0D // Odtwarzaj
17 #define DFPLAYER_CMD_PAUSE 0x0E // Pauza
18 #define DFPLAYER_CMD_STOP 0x16 // Stop
19 #define DFPLAYER_CMD_SPECIFY_FOLDER_PLAY 0x0F // Odtwórz plik z konkretnego folderu np. 01/001.mp3
20 #define DFPLAYER_CMD_QUERY_STATUS 0x42 // Zapytaj o status
21 #define DFPLAYER_CMD_QUERY_VOLUME 0x43 // Zapytaj o głośność
22 #define DFPLAYER_CMD_QUERY_FILES_COUNT 0x48 // Zapytaj o liczbę plików
23 #define DFPLAYER_CMD_SELECT_DEVICE 0x09 // Wybierz urządzenie (0x02 dla karty SD)
24
25 void dfplayer_init(UART_HandleTypeDef *huart, GPIO_TypeDef* busy_port, uint16_t busy_pin);
26 void dfplayer_send_cmd(uint8_t cmd, uint16_t arg);
27 void dfplayer_play_track(uint16_t track_num); // Odtwarza utwór numer X (w folderze domyślnym lub głównym)
28 void dfplayer_play_track_in_folder(uint8_t folder_num, uint8_t track_num_in_folder); // Odtwarza folder/plik
29 void dfplayer_set_volume(uint8_t volume); // 0-30
30 void dfplayer_next_track(void);
31 void dfplayer_prev_track(void);
32 void dfplayer_pause(void);
33 void dfplayer_resume(void);
34 void dfplayer_stop(void);
35 uint8_t dfplayer_is_busy(void);
36
37 #endif /* INC_DFPLAYER_MINI_H_ */
38
```





robota

Oliwier Bogdański, Kacper Szponar

## Dfplayer\_mini.c

```
1  /*
2   * dfplayer_mini.c
3   */
4
5
6  #include "dfplayer_mini.h"
7  #include <stdio.h>
8
9  static UART_HandleTypeDef *dfp_huart;
10 static GPIO_TypeDef* dfp_busy_port;
11 static uint16_t dfp_busy_pin;
12
13 // Bufor na komendę
14 static uint8_t cmd_buffer[10];
15
16 // Funkcja do obliczania sumy kontrolnej
17 static uint16_t calculate_checksum(uint8_t *buffer) {
18     uint16_t sum = 0;
19     for (int i = 1; i < 7; i++) { // Sumowanie od Version do Data_LSB
20         sum += buffer[i];
21     }
22     return -sum;
23 }
24
25 void dfplayer_init(UART_HandleTypeDef *huart, GPIO_TypeDef* busy_port, uint16_t busy_pin) {
26     dfp_huart = huart;
27     dfp_busy_port = busy_port;
28     dfp_busy_pin = busy_pin;
29
30     HAL_Delay(1000); // Czas na inicjalizację
31
32     // Wybranie karty SD jako źródło
33     dfplayer_send_cmd(DFPLAYER_CMD_SELECT_DEVICE, 0x0002); // 0x02 dla SD
34     HAL_Delay(200);
35
36     // Domyślna głośność
37     dfplayer_set_volume(10);
38     HAL_Delay(200);
39 }
40
```



robota

Oliwier Bogdański, Kacper Szponar

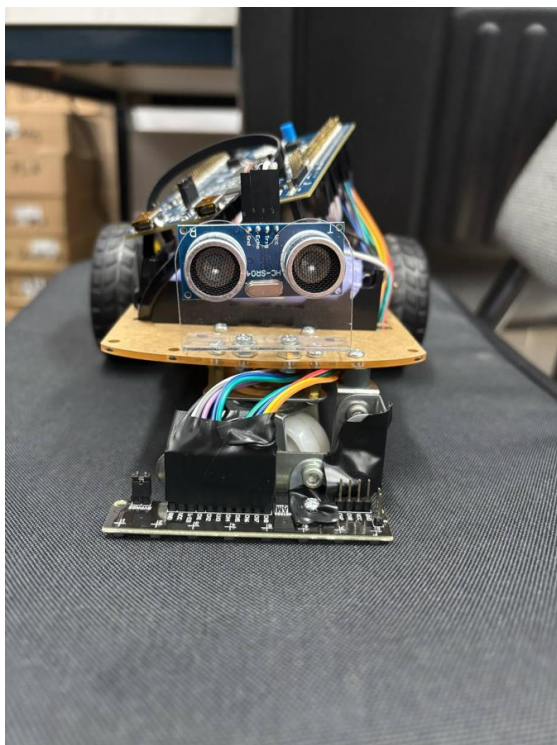
```
40
41 void dfplayer_send_cmd(uint8_t cmd_code, uint16_t arg) {
42     cmd_buffer[0] = 0x7E; // Bajt startu
43     cmd_buffer[1] = 0xFF; // Wersja
44     cmd_buffer[2] = 0x06; // Długość (nie licząc Start, End, Checksum)
45     cmd_buffer[3] = cmd_code; // Command
46     cmd_buffer[4] = 0x00; // Feedback (0x01 - z feedbackiem, 0x00 - bez)
47     cmd_buffer[5] = (uint8_t)(arg >> 8); // Argument MSB
48     cmd_buffer[6] = (uint8_t)(arg & 0xFF); // Argument LSB
49
50     uint16_t checksum = calculate_checksum(cmd_buffer);
51     cmd_buffer[7] = (uint8_t)(checksum >> 8); // Suma kontrolna MSB
52     cmd_buffer[8] = (uint8_t)(checksum & 0xFF); // Suma kontrolna LSB
53     cmd_buffer[9] = 0xEF; // Bajt końcowy
54
55     HAL_UART_Transmit(dfplayer_huart, cmd_buffer, 10, HAL_MAX_DELAY);
56 }
57
58 void dfplayer_play_track(uint16_t track_num) {
59     dfplayer_send_cmd(DFPLAYER_CMD_PLAY_TRACK, track_num);
60 }
61
62 void dfplayer_play_track_in_folder(uint8_t folder_num, uint8_t track_num_in_folder) {
63     uint16_t arg = ((uint16_t)folder_num << 8) | track_num_in_folder;
64     dfplayer_send_cmd(DFPLAYER_CMD_SPECIFY_FOLDER_PLAY, arg);
65 }
66
67 void dfplayer_set_volume(uint8_t volume) {
68     if (volume > 30) volume = 30;
69     dfplayer_send_cmd(DFPLAYER_CMD_SET_VOLUME, volume);
70 }
71
72 void dfplayer_next_track(void) {
73     dfplayer_send_cmd(DFPLAYER_CMD_NEXT_TRACK, 0);
74 }
75
76 void dfplayer_prev_track(void) {
77     dfplayer_send_cmd(DFPLAYER_CMD_PREV_TRACK, 0);
78 }
79
80 void dfplayer_pause(void) {
81     dfplayer_send_cmd(DFPLAYER_CMD_PAUSE, 0);
82 }
83
84 void dfplayer_resume(void) {
85     dfplayer_send_cmd(DFPLAYER_CMD_PLAY, 0);
86 }
87
88 void dfplayer_stop(void) {
89     dfplayer_send_cmd(DFPLAYER_CMD_STOP, 0);
90 }
91
```

robota

Oliwier Bogdański, Kacper Szponar

```
79  
80 void dfplayer_pause(void) {  
81     dfplayer_send_cmd(DFPLAYER_CMD_PAUSE, 0);  
82 }  
83  
84 void dfplayer_resume(void) {  
85     dfplayer_send_cmd(DFPLAYER_CMD_PLAY, 0);  
86 }  
87  
88 void dfplayer_stop(void) {  
89     dfplayer_send_cmd(DFPLAYER_CMD_STOP, 0);  
90 }  
91  
92 // Funkcja pomocnicza sprawdzająca stan logiczny pinu BUSY  
93 // BUSY pin jest LOW gdy odtwarza, HIGH gdy jest wolny/zatrzymany  
94 uint8_t dfplayer_is_busy(void) {  
95     if (GPIOC == NULL) return 0;  
96     return (HAL_GPIO_ReadPin(GPIOC, GPIO_PIN_13) == GPIO_PIN_RESET); // Zwraca 1 jeśli gra, 0 jeśli wolny  
97 }  
98  
<
```

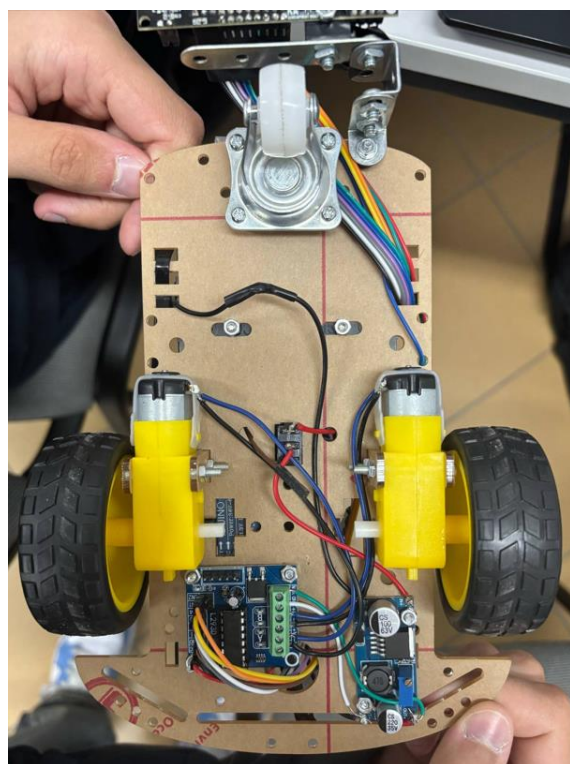
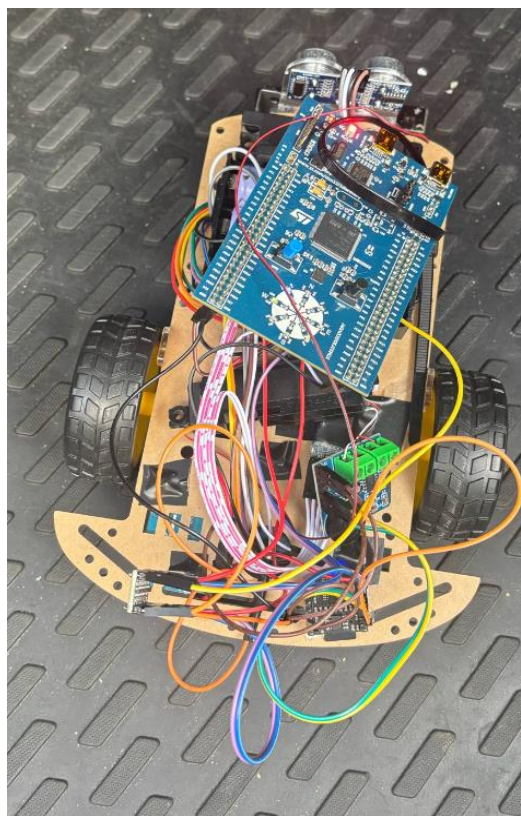
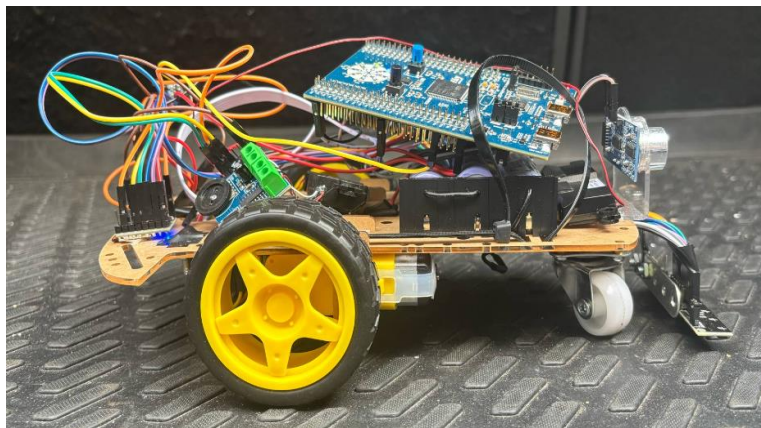
## 6. Zdjęcia opracowanego robota





robota

Oliwier Bogdański, Kacper Szponar





robotu

Oliwier Bogdański, Kacper Szponar

## 7. Wnioski

Projekt budowy robota został zrealizowany zgodnie z założeniami. Pojazd oparty na mikrokontrolerze STM32F303VCT6 Discovery porusza się autonomicznie po wyznaczonej linii, potrafi wykrywać przeszkody oraz odtwarzać dźwięki. Komunikacja za pomocą modułu Bluetooth umożliwia pełne sterowanie robotem, w tym kontrolę odtwarzania dźwięków przez DFPlayer, przełączanie trybów pracy (manualny/autonomiczny) oraz bezpośrednie sterowanie ruchem. Zarówno konstrukcja mechaniczna, jak i układ zasilania okazały się odpowiednie i zapewniły stabilną pracę urządzenia. Oprogramowanie zostało podzielone na moduły, co pozwoliło na przejrzyste zaimplementowanie wszystkich funkcji. W trakcie realizacji projektu udało się połączyć zagadnienia z zakresu elektroniki, programowania systemów wbudowanych oraz mechaniki. Wszystkie zaplanowane funkcjonalności zostały poprawnie zaimplementowane i przetestowane w praktyce.