AKADEMIA NAUK STOSOWANYCH w ELBLĄGU

Systemy Wbudowane i Mikroprocesory 2024/2025

Raport z budowy robota

Oliwier Bogdański, Kacper Szponar

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

Film prezentujący działanie robota: https://youtube.com/shorts/g8iPMdD_pI8

1. Opis robota

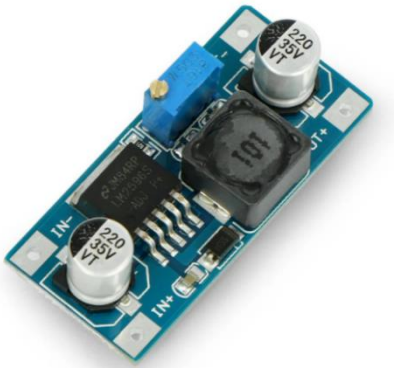
Robot to dwukołowy autonomiczny pojazd mobilny z jednym dodatkowym kołem obrotowym dla stabilizacji, oparty na mikrokontrolerze STM32F303VCT6 Discovery. Napęd realizowany jest przez dwa silniki elektryczne z przekładniami sterowane za pomocą sygnałów PWM oraz sterownika silników L293D. Skręt wykonywany jest różnicowo – poprzez zmianę prędkości obrotowej poszczególnych silników. Układ zasilany jest autonomicznie przez trzy oginwa 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. Dodatkowo, do wykrywania przeszkód na drodze wykorzystano czujnik ultradźwiękowy HC-SR04, który mierzy odległość od obiektów na podstawie czasu powrotu fali ultradźwiękowej odbitej od przeszkody. Szacowany rozmiar robota to około 200 x 140 x 65 mm. Planowana dodatkowa funkcjonalność to dodanie głośnika.

2. Elementy wybrane do budowy robota

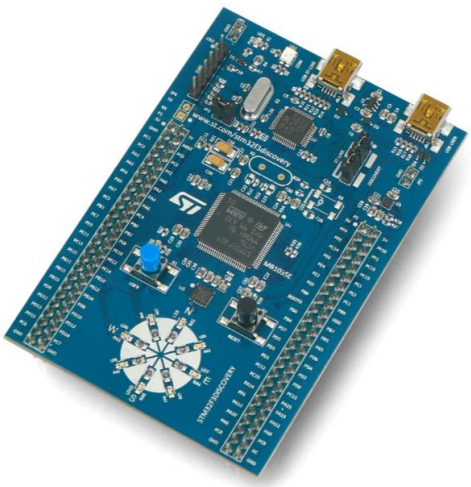
Podwozie: Chassis Rectangle 2WD 2-kołowe podwozie robota, oraz obrotowe koło podporowe

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

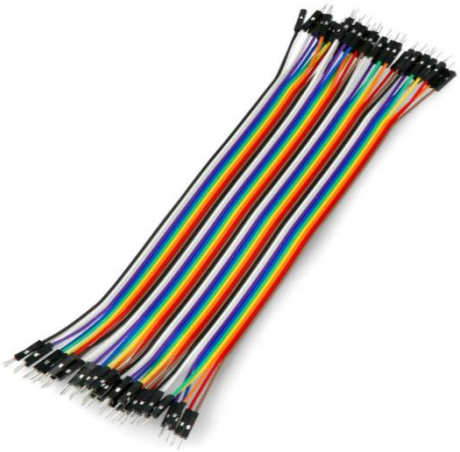
Zasilanie: 3x Ogniwo 18650 Li-Ion INR18650-F1HR 3350mAh, koszyk na 3 akumulatory typu 18650 oraz przetwornica napięcia LM2596HVS



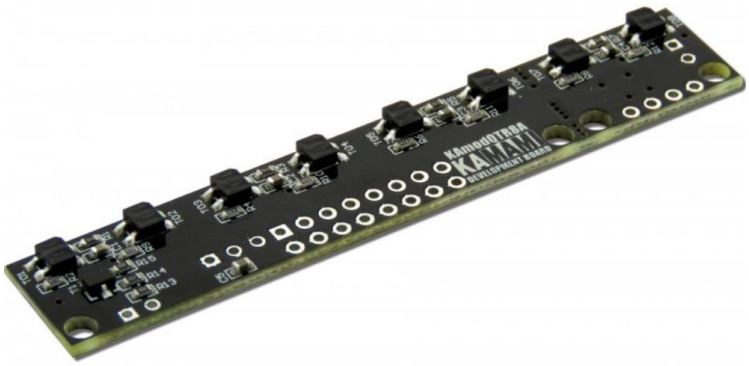
Sterowanie: Mikrokontroler STM32F303VCT6 (Discovery), sterownik silnika krokowego 2 DC L293D mini mostek H



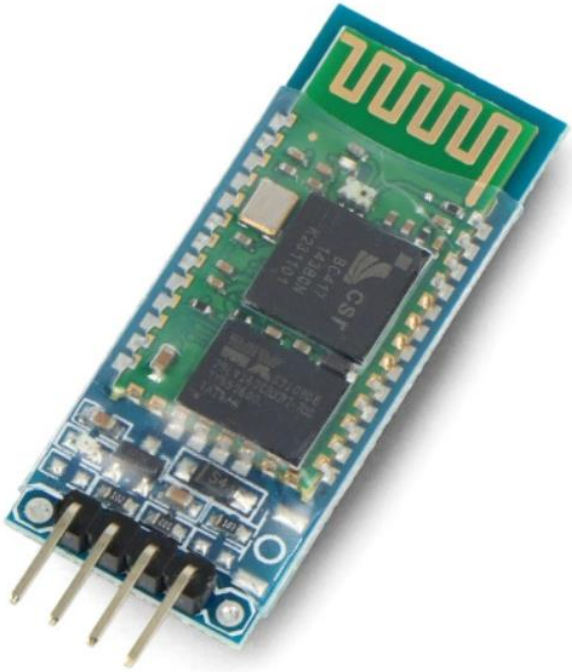
Elementy montażowe: Śrubki o rozmiarze m3, przewody połączeniowe.



Czujniki: KAmodQTR8A - moduł z ośmioma czujnikami odbiciowymi KTIR0711S (wykrycie linii), Ultradźwiękowy czujnik odległości HC-SR04 (wykrycie przeszkody).



Komunikacja: Moduł Bluetooth HC-05

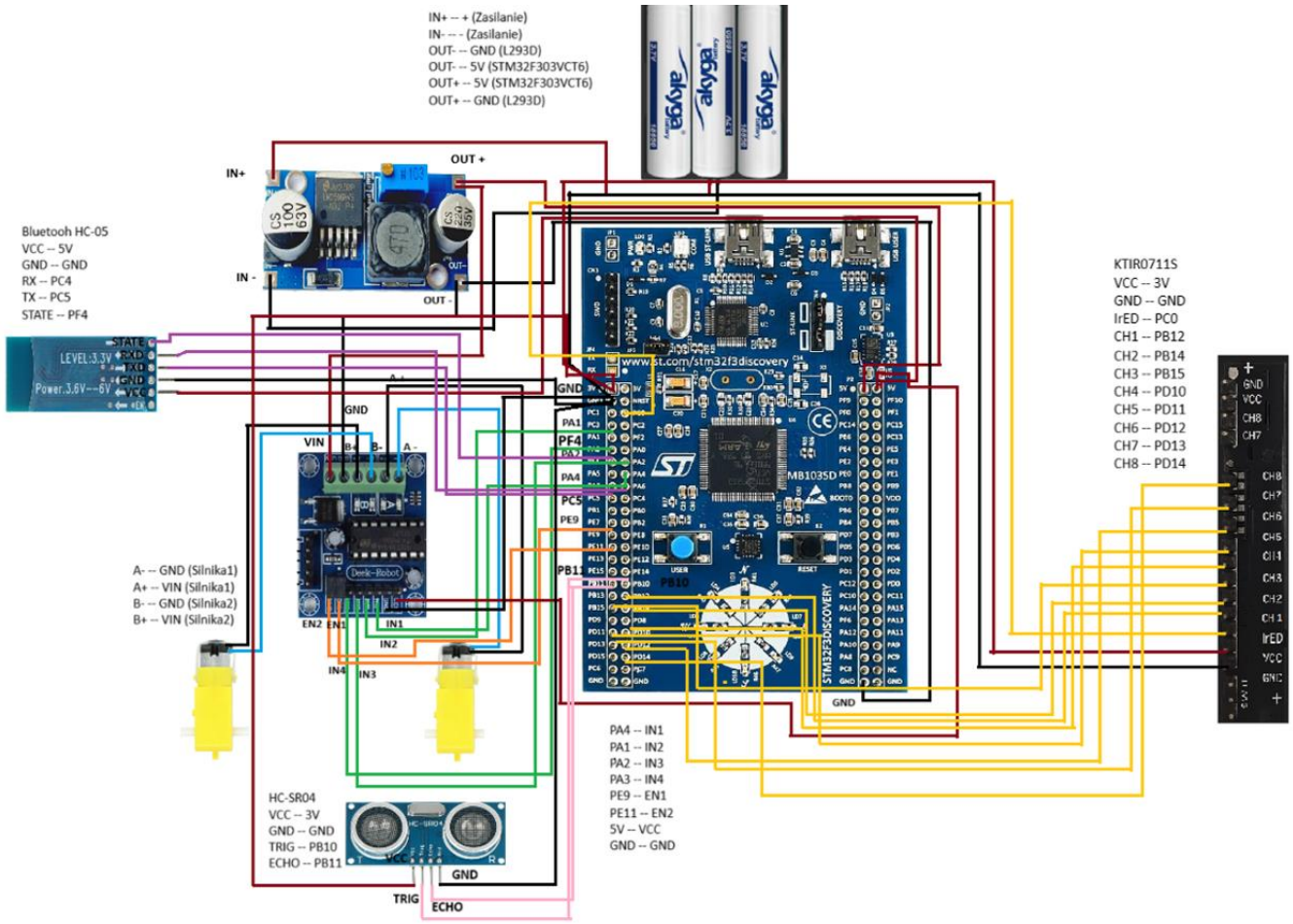


---

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.

4. Schemat elektroniczny robota



5. Oprogramowanie sterujące

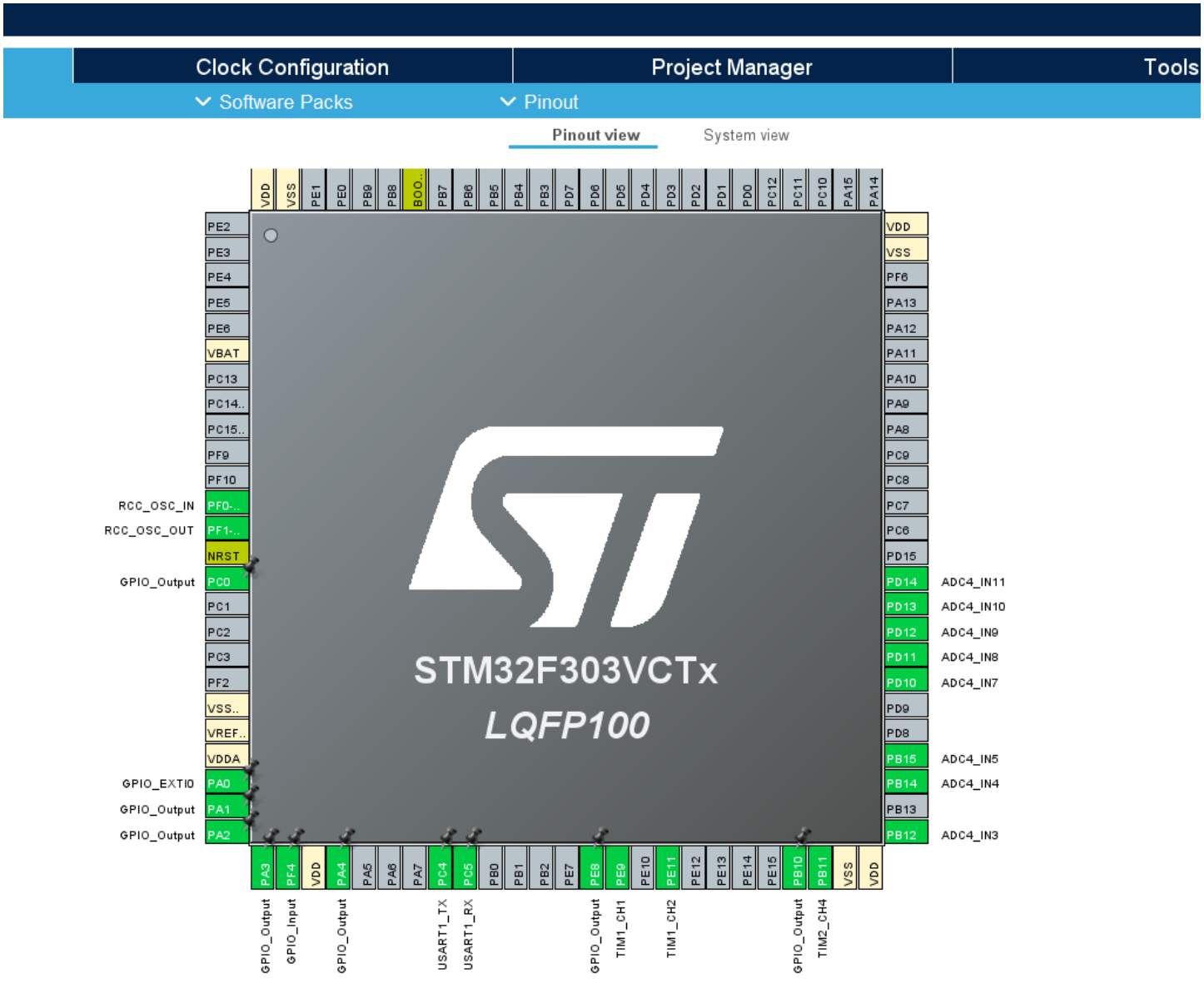
// Najważniejsze fragmenty kodu źródłowego wraz z opisem

robota

Oliwier Bogdański, Kacper Szponar

Konfiguracja ioc

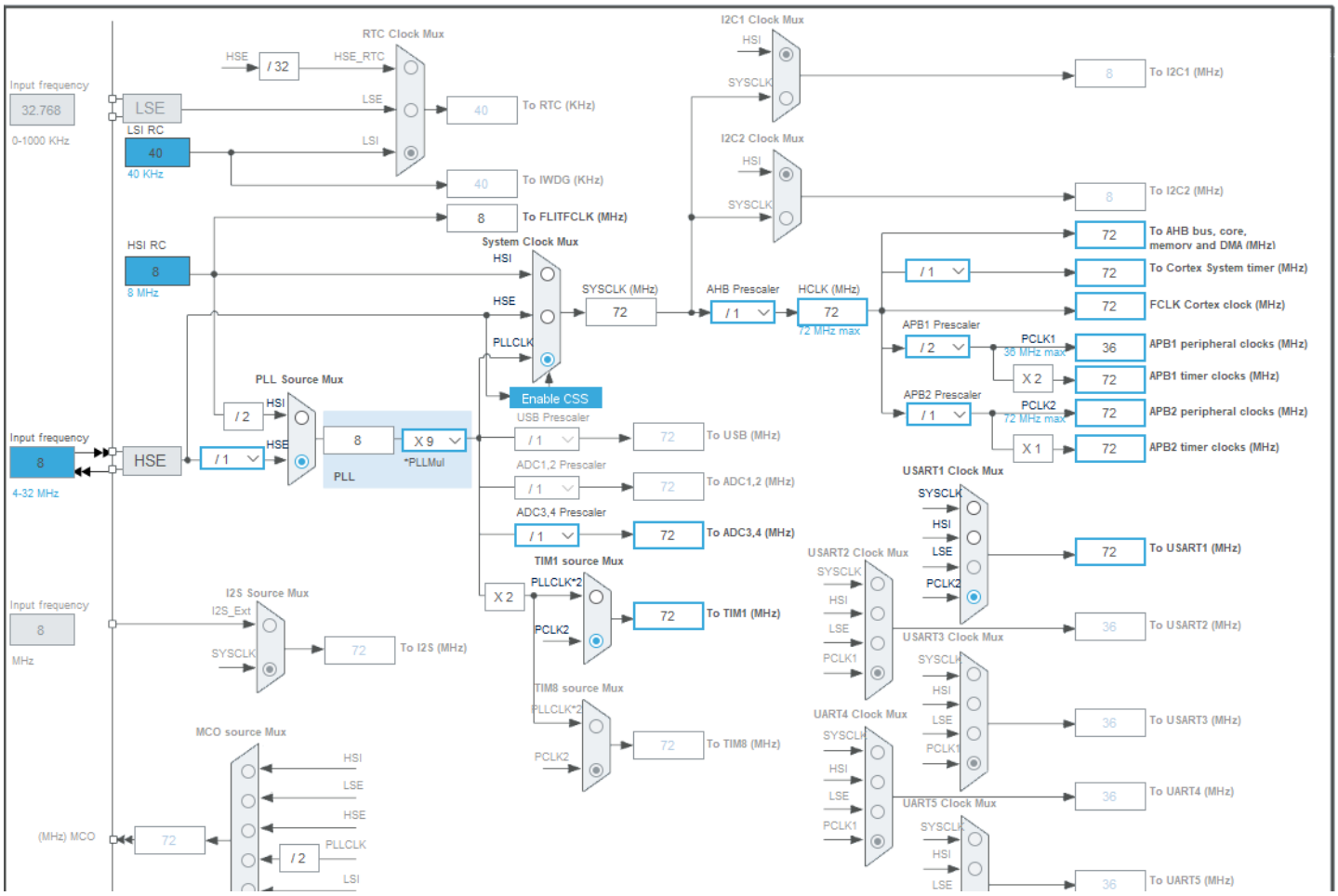
Pinout



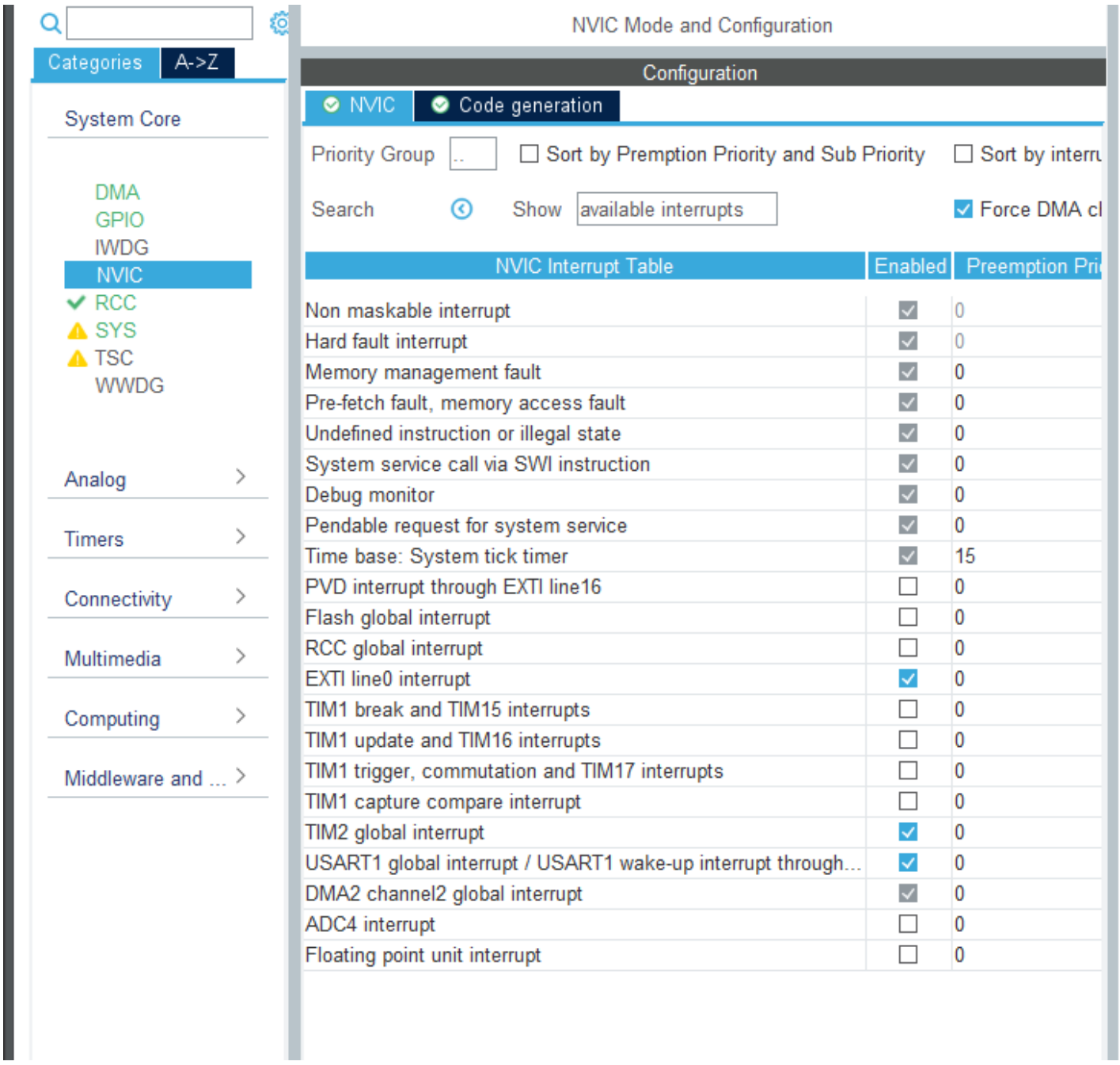
robota

Oliwier Bogdański, Kacper Szponar

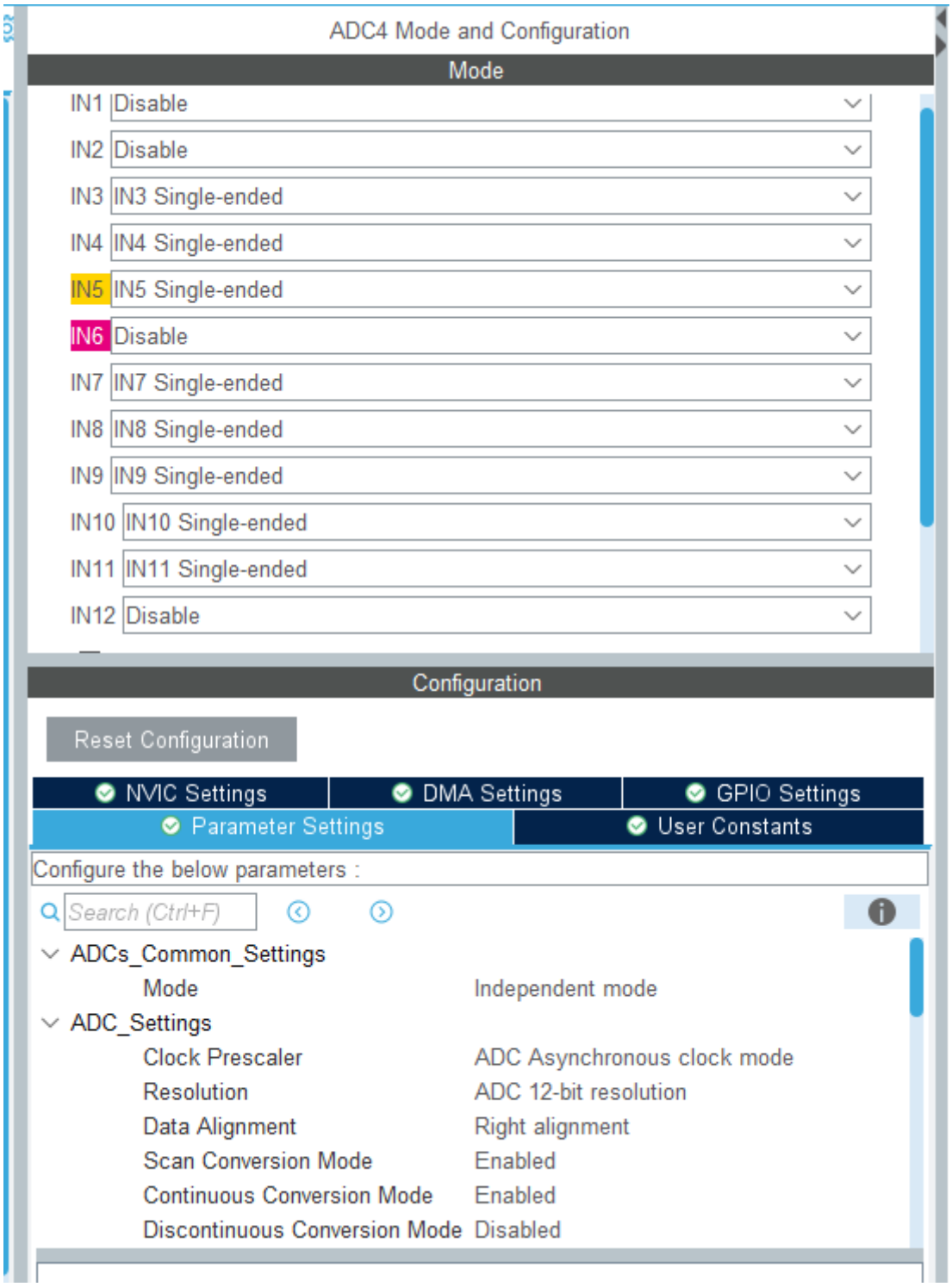
CLOCK



NVIC


NVIC Mode and Configuration
Configuration
NVIC
Code generation
Priority Group
Sort by Premption Priority and Sub Priority
Search
Show
available interrupts
Force DMA cl
Categories
A->Z
System Core
DMA
GPIO
IWDG
NVIC
RCC
SYS
TSC
WWDG
Analog
Timers
Connectivity
Multimedia
Computing
Middleware and ...
NVIC Interrupt Table
Enabled
Preemption Pri
Non maskable interrupt 0
Hard fault interrupt 0
Memory management fault 0
Pre-fetch fault, memory access fault 0
Undefined instruction or illegal state 0
System service call via SWI instruction 0
Debug monitor 0
Pendable request for system service 0
Time base: System tick timer 15
PVD interrupt through EXTI line16 0
Flash global interrupt 0
RCC global interrupt 0
EXTI line0 interrupt 0
TIM1 break and TIM15 interrupts 0
TIM1 update and TIM16 interrupts 0
TIM1 trigger, commutation and TIM17 interrupts 0
TIM1 capture compare interrupt 0
TIM2 global interrupt 0
USART1 global interrupt / USART1 wake-up interrupt through... 0
DMA2 channel2 global interrupt 0
ADC4 interrupt 0
Floating point unit interrupt 0

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)
ADCs_Common_Settings
Mode Independent mode
ADC_Settings
Clock Prescaler ADC Asynchronous clock mode
Resolution ADC 12-bit resolution
Data Alignment Right alignment
Scan Conversion Mode Enabled
Continuous Conversion Mode Enabled
Discontinuous Conversion Mode Disabled

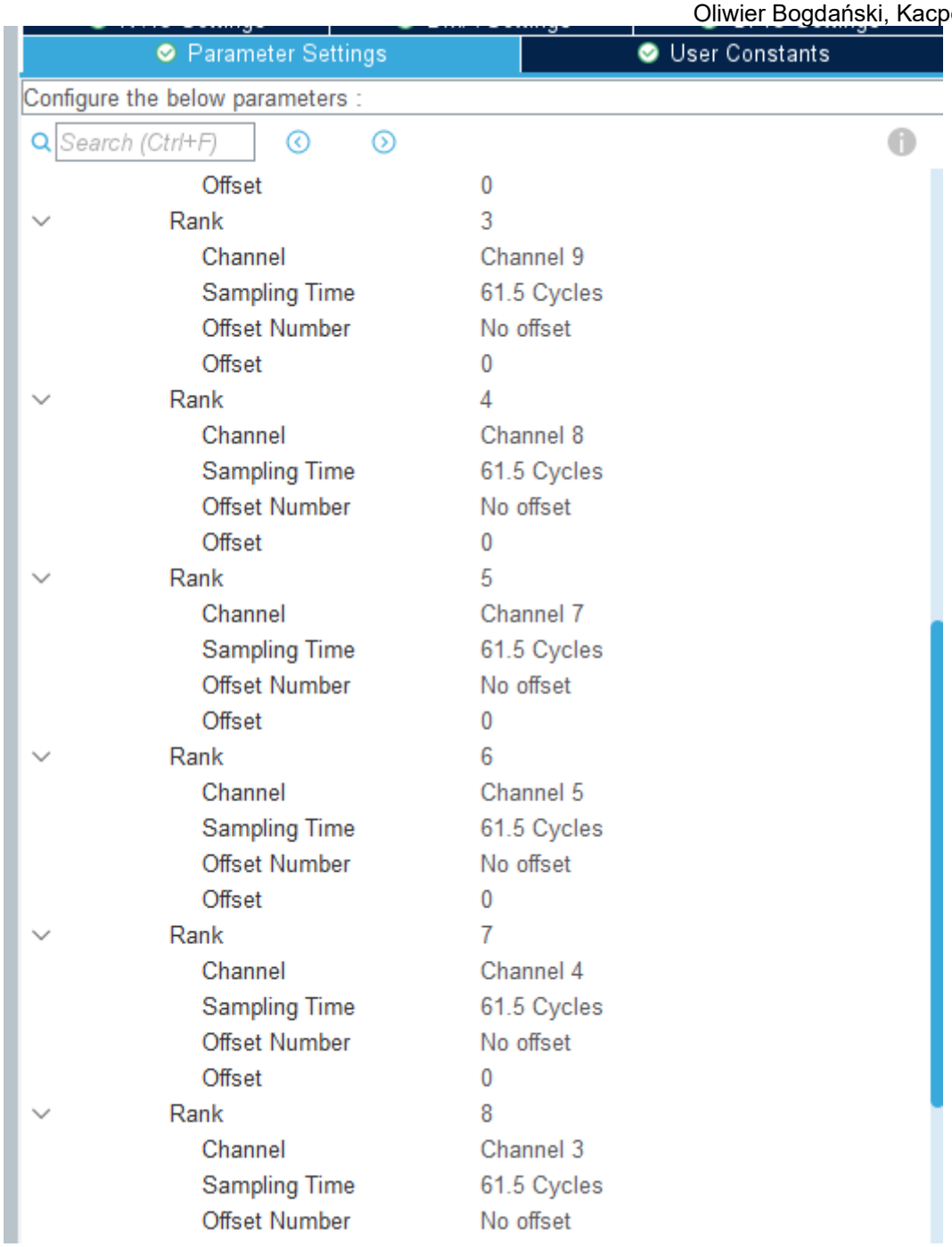
NVIC Settings | DMA Settings | GPIO Settings

Parameter Settings | User Constants

Configure the below parameters :

Search (Ctrl+F)

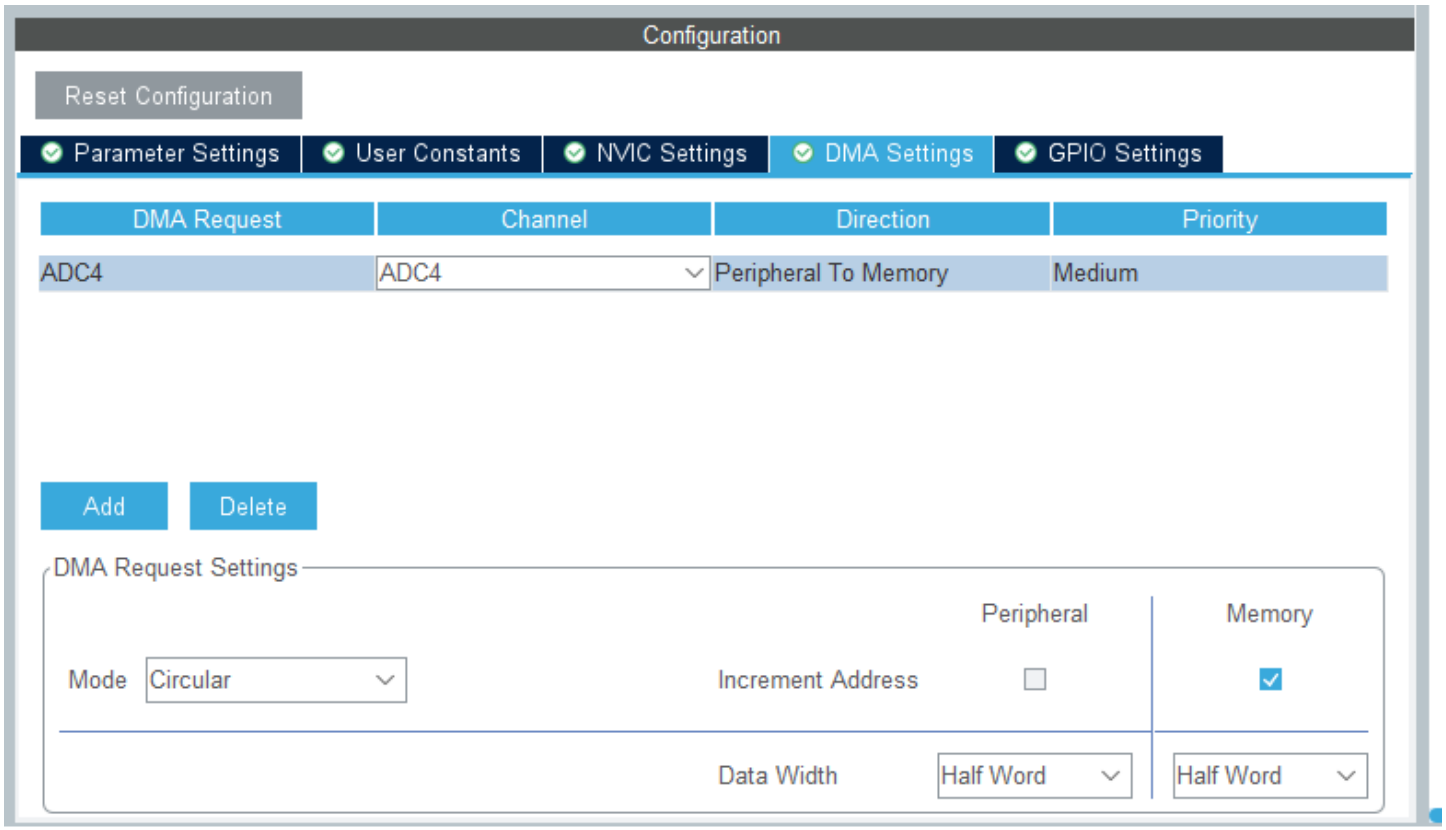
ADCs_Common_Settings	
Mode	Independent mode
ADC_Settings	
Clock Prescaler	ADC Asynchronous clock mode
Resolution	ADC 12-bit resolution
Data Alignment	Right alignment
Scan Conversion Mode	Enabled
Continuous Conversion Mode	Enabled
Discontinuous Conversion Mode	Disabled
DMA Continuous Requests	Enabled
End Of Conversion Selection	End of single conversion
Overrun behaviour	Overrun data overwritten
Low Power Auto Wait	Disabled
ADC_Regular_ConversionMode	
Enable Regular Conversions	Enable
Number Of Conversion	8
External Trigger Conversion S...	Regular Conversion launched by software
External Trigger Conversion E...	None
SequencerNbRanks	1
Rank	1
Channel	Channel 11
Sampling Time	61.5 Cycles
Offset Number	No offset
Offset	0
Rank	2
Channel	Channel 10
Sampling Time	61.5 Cycles
Offset Number	No offset


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

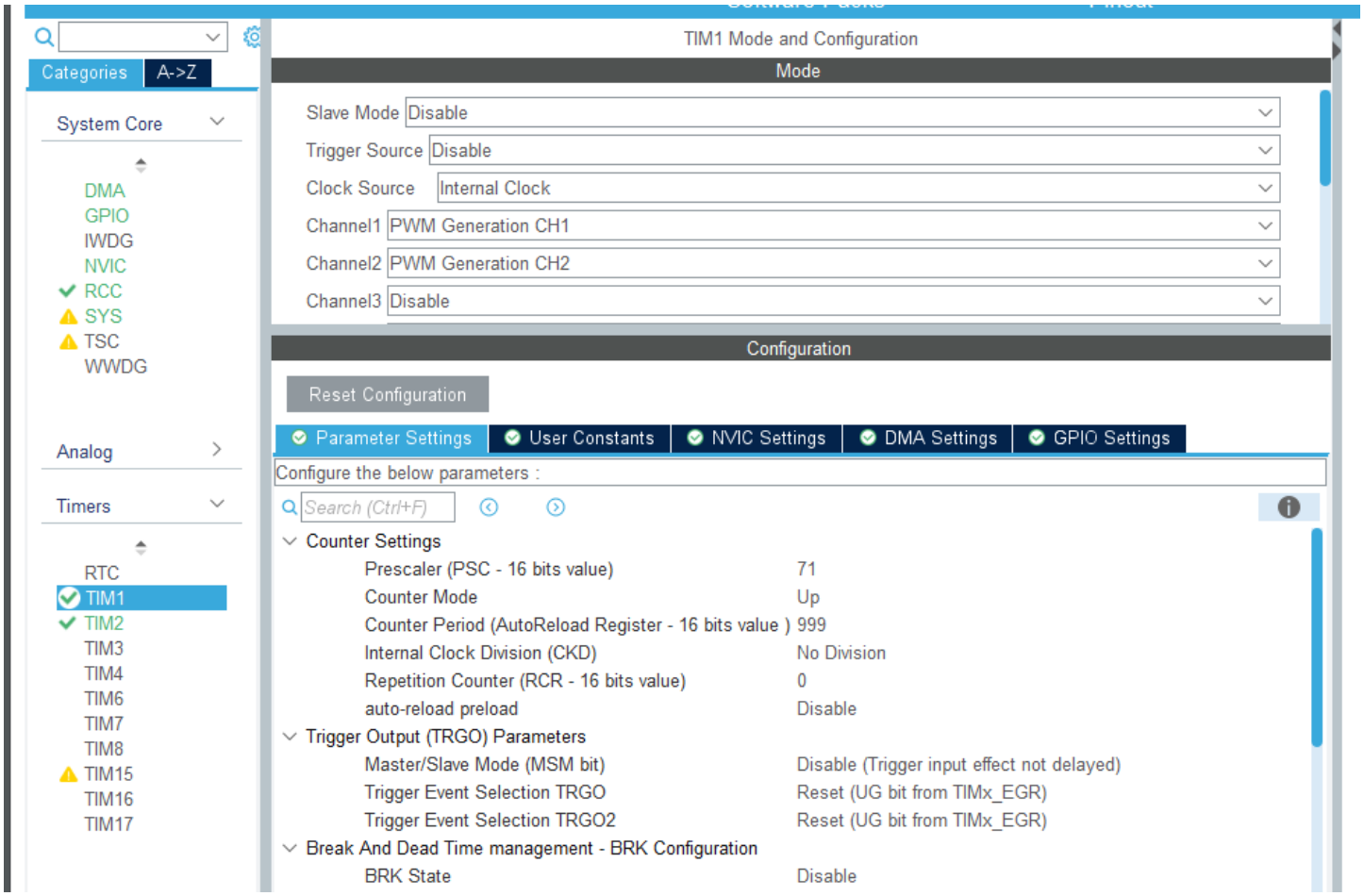
robota

Oliwier Bogdański, Kacper Szponar

DMA



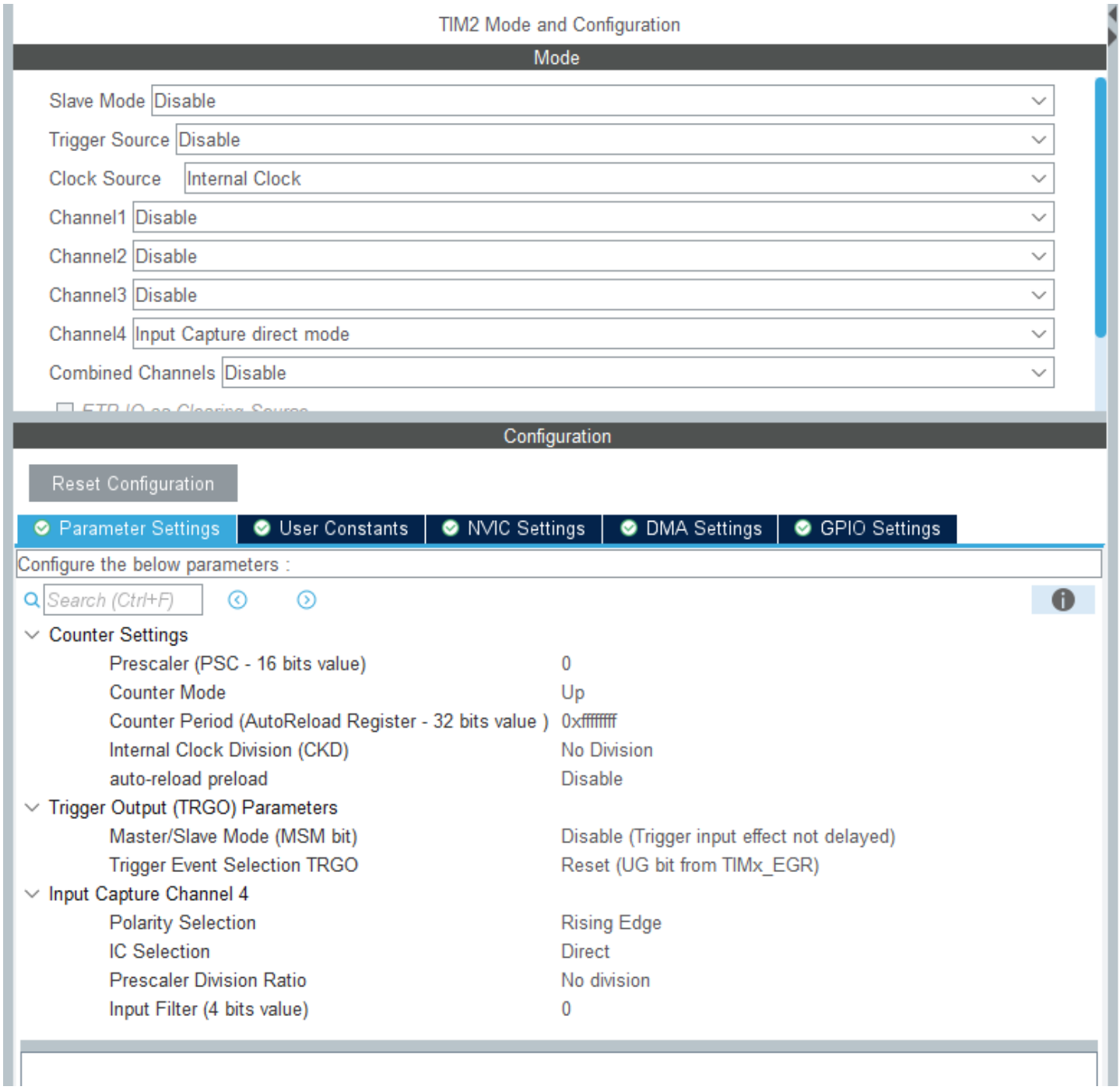
TIM1


TIM1 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
Mode
Slave Mode Disable
Trigger Source Disable
Clock Source Internal Clock
Channel1 PWM Generation CH1
Channel2 PWM Generation CH2
Channel3 Disable
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

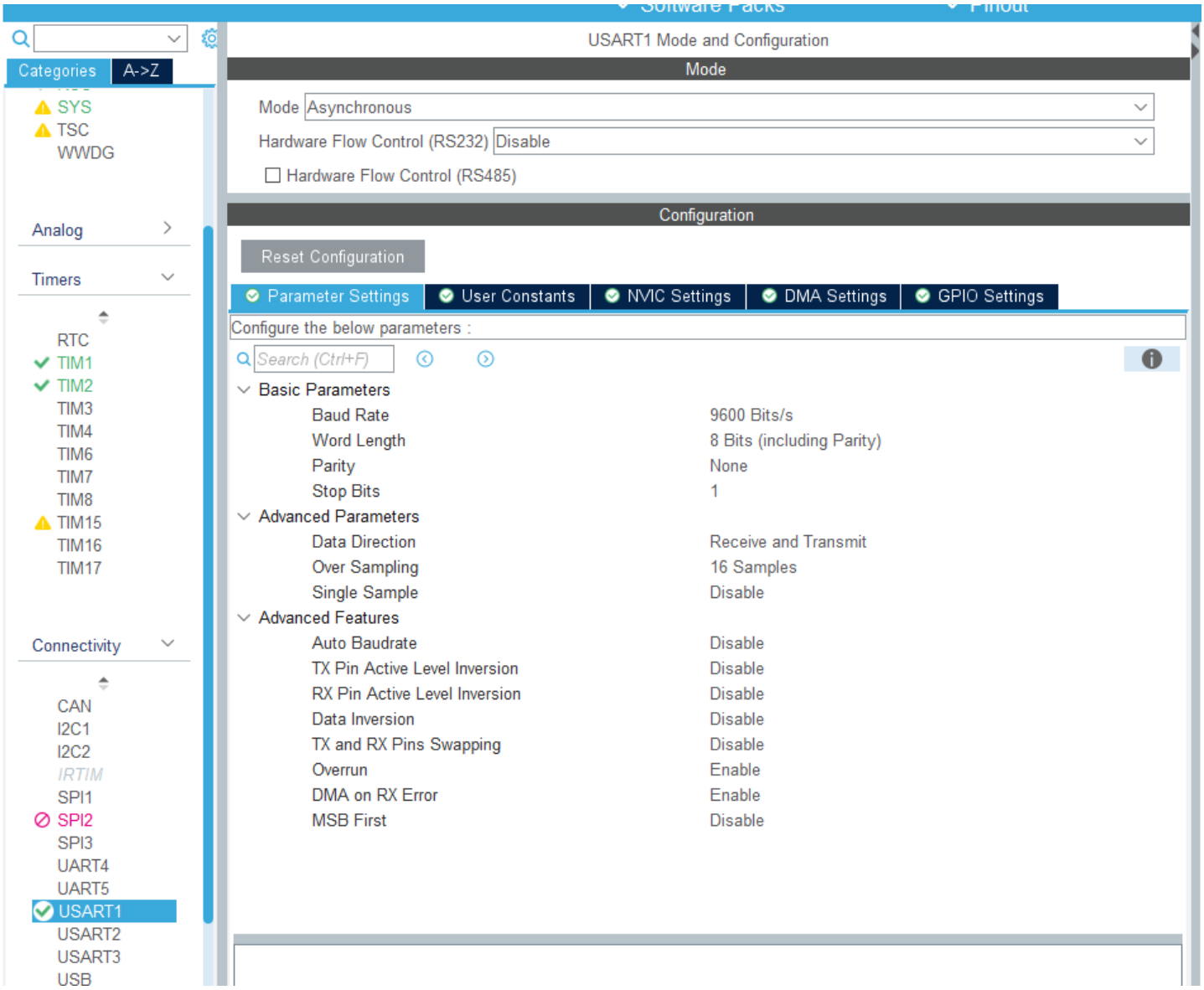
robota

Oliwier Bogdański, Kacper Szponar

TIM2



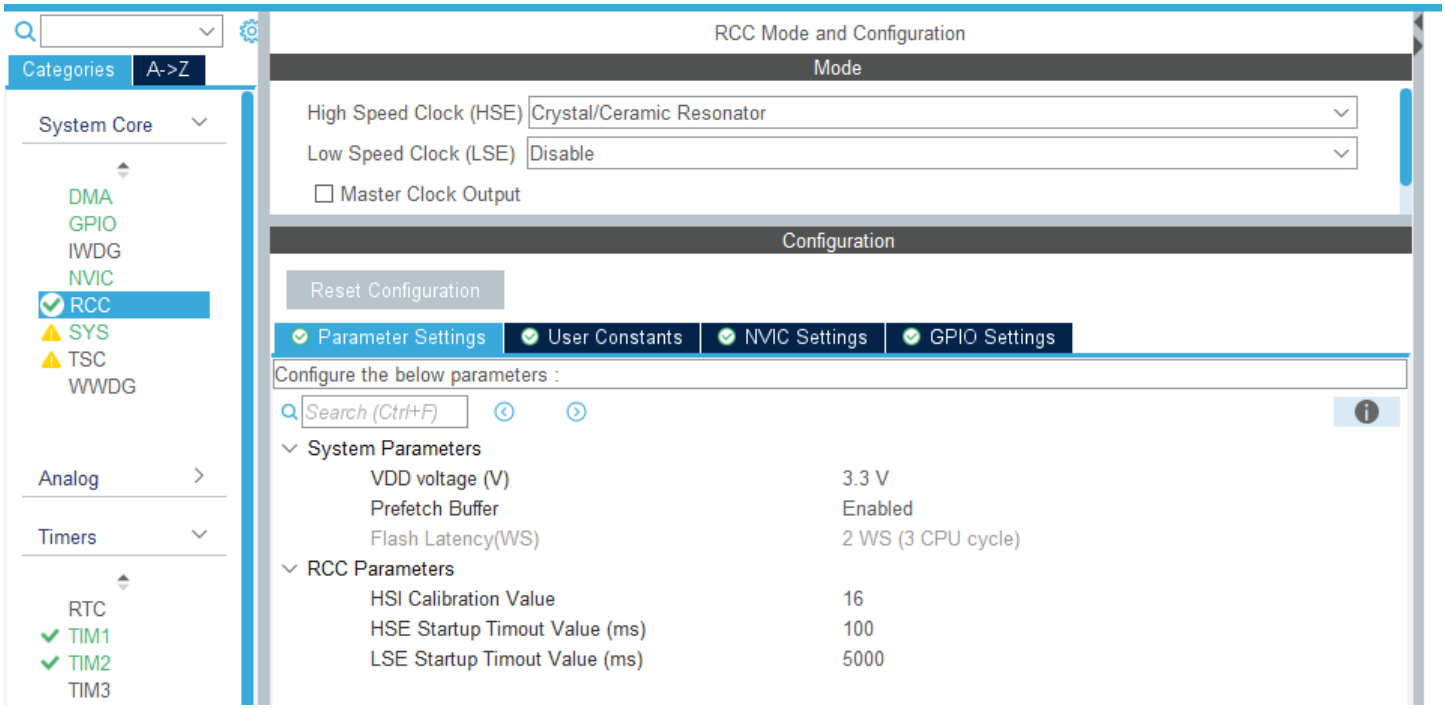
USART1


Software Packs
Pinout
USART1 Mode and Configuration
Mode
Mode Asynchronous
Hardware Flow Control (RS232) Disable
Hardware Flow Control (RS485)
Configuration
Reset Configuration
Parameter Settings
User Constants
NVIC Settings
DMA Settings
GPIO Settings
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 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
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
USART1
USART2
USART3
USB

robota

Oliwier Bogdański, Kacper Szponar

RCC



Oliwier Bogdański, Kacper Szponar

Kod Programu

Main.c

```c
#include "main.h"

/* Private includes ----------------------------------------------------------*/
/* USER CODE BEGIN Includes */
#include "line_follower.h"
#include <string.h> // Potrzebne dla strlen
#include <stdio.h>
#include <ctype.h>
/* USER CODE END Includes */

/* Private typedef -----------------------------------------------------------*/
/* USER CODE BEGIN PTD */

/* USER CODE END PTD */

/* Private define ------------------------------------------------------------*/
/* USER CODE BEGIN PD */

// Typ enum dla trybów pracy robota
typedef enum {
    ROBOT_MODE_STOPPED,
    ROBOT_MODE_LINE_FOLLOWER,
    ROBOT_MODE_BLUETOOTH_MANUAL
} RobotOperatingMode;

#define ADC_CHANNELS 8 // Definicja liczby kanałów ADC używanych do odczytu czujników linii

uint16_t adc_buffer[ADC_CHANNELS]; // Bufor przechowujący odczyty z czujników linii (ADC)

volatile uint8_t robot_running = 0; // Zmienna globalna ustawiana w przerwaniu po naciśnięciu przycisku użytkownika (USER button).

volatile uint32_t echo_start = 0;    // Czas (w tickach timera) rozpoczęcia odbieranego echa
volatile uint32_t echo_end = 0;      // Czas (w tickach timera) zakończenia odbieranego echa
volatile uint8_t echo_captured = 0; // Flaga statusu przechwytywania echa (0: nic, 1: start, 2: koniec).

volatile uint8_t uart_rx_buffer; // Bufor na odebrany bajt z UART (Bluetooth)
const uint16_t BT_PWM_SPEED_RIGHT = 900; // Prędkość dla prawego silnika przy sterowaniu BT
const uint16_t BT_PWM_SPEED_LEFT  = 999; // Prędkość dla lewego silnika przy sterowaniu BT
/* USER CODE END PD */

```

```c
void robot_drive(uint16_t pwm_right, uint8_t dir_right,
                 uint16_t pwm_left, uint8_t dir_left)
{
    __HAL_TIM_SET_COMPARE(&htim1, TIM_CHANNEL_1, pwm_right); // Ustawia wypełnienie PWM na kanale 1 (prawy silnik) - kontrola prędkości
    __HAL_TIM_SET_COMPARE(&htim1, TIM_CHANNEL_2, pwm_left); // Ustawia wypełnienie PWM na kanale 2 (lewy silnik) - kontrola prędkości
    // ustawianie odpowiednich stanów logicznych na wyjściach GPIO
    // prawy silnik
    if (dir_right) {
        HAL_GPIO_WritePin(GPIOA, GPIO_PIN_4, GPIO_PIN_SET);   // IN1
        HAL_GPIO_WritePin(GPIOA, GPIO_PIN_1, GPIO_PIN_RESET); // IN2
    } else {
        HAL_GPIO_WritePin(GPIOA, GPIO_PIN_4, GPIO_PIN_RESET); // IN1
        HAL_GPIO_WritePin(GPIOA, GPIO_PIN_1, GPIO_PIN_SET);   // IN2
    }

    // lewy silnik
    if (dir_left) {
        HAL_GPIO_WritePin(GPIOA, GPIO_PIN_2, GPIO_PIN_SET);   // IN3
        HAL_GPIO_WritePin(GPIOA, GPIO_PIN_3, GPIO_PIN_RESET); // IN4
    } else {
        HAL_GPIO_WritePin(GPIOA, GPIO_PIN_2, GPIO_PIN_RESET); // IN3
        HAL_GPIO_WritePin(GPIOA, GPIO_PIN_3, GPIO_PIN_SET);   // IN4
    }

}

// Funkcja zatrzymująca ruch robota. Ustawia wszystkie wyjścia GPIO NA 0 oraz wypełnienie PWM na 0 na obu silnikach
void robot_stop()
{
    HAL_GPIO_WritePin(GPIOA, GPIO_PIN_4, GPIO_PIN_RESET);
    HAL_GPIO_WritePin(GPIOA, GPIO_PIN_1, GPIO_PIN_RESET);
    HAL_GPIO_WritePin(GPIOA, GPIO_PIN_2, GPIO_PIN_RESET);
    HAL_GPIO_WritePin(GPIOA, GPIO_PIN_3, GPIO_PIN_RESET);
    __HAL_TIM_SET_COMPARE(&htim1, TIM_CHANNEL_1, 0);
    __HAL_TIM_SET_COMPARE(&htim1, TIM_CHANNEL_2, 0);
}

// Funkcja Generuje krótki (10µs) impuls na pinie TRIG czujnika HC-SR04, aby zainicjować pomiar odległości
void HCSR04_Trigger(void)
{
    // Ustawia pin PB10 (TRIG) w stan wysoki, czeka 10 mikrosekund, a następnie ustawia go w stan niski.
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_10, GPIO_PIN_SET);
    HAL_Delay(0.01);  // 10 us za pomocą HAL_Delay. Do zmiany w przyszłości aby liczyło czas za pomocą timera.
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_10, GPIO_PIN_RESET);
}
```

```c
// Funkcja mierząca odległość za pomocą czujnika HC-SR04
// Inicjuje pomiar, czeka na impuls echa, oblicza czas jego trwania i przelicza na odległość w cm
// Używa timera TIM2 w trybie Input Capture do pomiaru czasu trwania impulsu echa
float HCSR04_ReadDistance(void)
{
    echo_captured = 0; // Reset flagi statusu przechwytywania echa

    // Uruchamia timer TIM2 w trybie Input Capture na kanale 4 (oczekiwanie na zbocze narastające)
    HAL_TIM_IC_Start_IT(&htim2, TIM_CHANNEL_4);

    HCSR04_Trigger(); // Wywołanie funkcji wysyłającej impuls

    uint32_t timeout = HAL_GetTick() + 100; // timeout na 100ms
    while (echo_captured < 2) // Czeka, aż zostaną przechwycone oba zbocza impulsu echa (start i end)
    {
        if (HAL_GetTick() > timeout)
        {
            return -1.0f; // wystąpił timeout (brak echa w ciągu 100ms), funkcja zwraca błąd -1.0f
        }
    }
    // Zatrzymanie timera
    HAL_TIM_IC_Stop_IT(&htim2, TIM_CHANNEL_4);

    uint32_t ticks; // Zmienna na czas trwania impulsu echa w tickach timera
    if (echo_end >= echo_start)          // Normalny przypadek
        ticks = echo_end - echo_start;
    else                                 // Przypadek przepełnienia licznika timera
        ticks = (0xFFFFFFFF - echo_start + echo_end);

    float distance = ticks * 0.000239463f;
    // Obliczony współczynnik dla zegara systemowego 72MHz i prędkosci dzwięku ok 343m/s
    // Daje on wystarczającą dokładność dla wykrywania przeszkód podczas jazdy oraz
    // mierzenia dystansu w zakresie pracy czujnika

    return distance; // Zwrócenie obliczonej odległości
}

// Funkcja pomocnicza do wysyłająca wiadomości poprzez UART
// Jako parametr przyjmuje wskaźnik do ciągu znaków do wysłania
void UART_SendString(char *str) {
    HAL_UART_Transmit(&huart1, (uint8_t*)str, strlen(str), HAL_MAX_DELAY);
}


// Funkcja sprawdzająca stan połączenia Bluetooth
uint8_t HC05_State(void) {
    // Zwraca 1 jeśli połączony (HC-05 wysyła HIGH na pinie State)
    return HAL_GPIO_ReadPin(GPIOF, GPIO_PIN_4);
}

```

Oliwier Bogdański, Kacper Szponar

```c
// Funkcja przetwarzająca komendy z Bluetooth sterując ruchem robota lub zmieniając jego tryb pracy.
// Wywoływana w callback do przerwania USART1
void ProcessBluetoothCommand(uint8_t command) {
    uint8_t cmd_lower = tolower(command); //normalizacja to lower case

    // Opcjonalne wysyłanie potwierdzenia odebrania wiadomości
    // snprintf(msg, sizeof(msg), "BT RX: %c\r\n", command);
    // UART_SendString(msg);

    switch (cmd_lower) {
        case 'f': // Przód
            if (current_robot_mode == ROBOT_MODE_BLUETOOTH_MANUAL) {
                robot_drive(BT_PWM_SPEED_RIGHT, 1, BT_PWM_SPEED_LEFT, 1);
                UART_SendString("BT CMD: FWD\r\n");
            } else {
                UART_SendString("BT Info: Tryb manualny nie aktywny\r\n");
            }
            break;
        case 'b': // Tył
            if (current_robot_mode == ROBOT_MODE_BLUETOOTH_MANUAL) {
                robot_drive(BT_PWM_SPEED_RIGHT, 0, BT_PWM_SPEED_LEFT, 0);
                UART_SendString("BT CMD: BCK\r\n");
            } else {
                UART_SendString("BT Info: Tryb manualny nie aktywny\r\n");
            }
            break;
        case 'l': // Lewo
            if (current_robot_mode == ROBOT_MODE_BLUETOOTH_MANUAL) {
                robot_drive(BT_PWM_SPEED_RIGHT, 1, BT_PWM_SPEED_LEFT, 0);
                UART_SendString("BT CMD: LEFT\r\n");
            } else {
                UART_SendString("BT Info: Tryb manualny nie aktywny\r\n");
            }
            break;
        case 'r': // Prawo
            if (current_robot_mode == ROBOT_MODE_BLUETOOTH_MANUAL) {
                robot_drive(BT_PWM_SPEED_RIGHT, 0, BT_PWM_SPEED_LEFT, 1);
                UART_SendString("BT CMD: RIGHT\r\n");
            } else {
                UART_SendString("BT Info: Tryb manualny nie aktywny\r\n");
            }
            break;
        case 's': // Stop
            robot_running = 0; // flaga na 0
            robot_stop();
            current_robot_mode = ROBOT_MODE_BLUETOOTH_MANUAL; // Po stop pozostaje w trybie manualnym aby płynnej sterować aplikacją w telefonie
            line_follower_reset_state();
            UART_SendString("BT CMD: STOP\r\n");
            break;

        case 'a': // Tryb autonomiczny - śledzenie linii
            if (current_robot_mode != ROBOT_MODE_LINE_FOLLOWER || !robot_running) { // Jeśli nie był już w LF i nie jechał
                robot_stop(); // Zatrzymanie, jeśli np. jedzie w trybie BT
```

```
            if (current_robot_mode == ROBOT_MODE_BLUETOOTH_MANUAL) {
                robot_drive(BT_PWM_SPEED_RIGHT, 1, BT_PWM_SPEED_LEFT, 0);
                UART_SendString("BT CMD: LEFT\r\n");
            } else {
                UART_SendString("BT Info: Tryb manualny nie aktywny\r\n");
            }
            break;
        case 'r': // Prawo
            if (current_robot_mode == ROBOT_MODE_BLUETOOTH_MANUAL) {
                robot_drive(BT_PWM_SPEED_RIGHT, 0, BT_PWM_SPEED_LEFT, 1);
                UART_SendString("BT CMD: RIGHT\r\n");
            } else {
                UART_SendString("BT Info: Tryb manualny nie aktywny\r\n");
            }
            break;
        case 's': // Stop
            robot_running = 0; // flaga na 0
            robot_stop();
            current_robot_mode = ROBOT_MODE_BLUETOOTH_MANUAL; // Po stop pozostaje w trybie manualnym aby płynnej sterować aplikacją w telefonie
            line_follower_reset_state();
            UART_SendString("BT CMD: STOP\r\n");
            break;

        case 'a': // Tryb autonomiczny - śledzenie linii
            if (current_robot_mode != ROBOT_MODE_LINE_FOLLOWER || !robot_running) { // Jeśli nie był już w LF i nie jechał
                robot_stop(); // Zatrzymanie, jeśli np. jedzie w trybie BT
            }
            current_robot_mode = ROBOT_MODE_LINE_FOLLOWER;
            robot_running = 1; // Odrazu jedzie (zmienna z PA0 ustawiona na 1)
            line_follower_reset_state();
            UART_SendString("BT CMD: Tryb -> Śledzenie liniir");
            break;

        case 'm': // Tryb manualny - sterowanie Bluetooth
            if (robot_running && current_robot_mode == ROBOT_MODE_LINE_FOLLOWER) {
                robot_stop(); // Zatrzymanie jeśli jedzie
            }
            current_robot_mode = ROBOT_MODE_BLUETOOTH_MANUAL;
            robot_running = 0; // W trybie manualnym, `robot_running` nie kontroluje bezpośrednio
            line_follower_reset_state();
            UART_SendString("BT CMD: Tryb -> Sterowanie ręczne\r\n");
            break;

            // MIEJSCE NA KOMENDĘ DO POMIARU DYSTANSU

        default:
            UART_SendString("BT CMD: Nieznana komenda\r\n");
            break;
    }
}
/* USER CODE END 0 */
```

Funkcja Main()

```c
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_DMA_Init();
  MX_TIM1_Init();
  MX_TIM2_Init();
  MX_USART1_UART_Init();
  MX_ADC4_Init();
  /* USER CODE BEGIN 2 */
  HAL_TIM_PWM_Start(&htim1, TIM_CHANNEL_1);// Start generowania sygnału PWM na kanale 1 (prawy silnik)
  HAL_TIM_PWM_Start(&htim1, TIM_CHANNEL_2); // Start generowania sygnału PWM na kanale 2 (lewy silnik)

  HAL_GPIO_WritePin(GPIOC, GPIO_PIN_0, GPIO_PIN_SET); // IrED = 1 -> Diody IR w czujniku odbiciowym włączone

  line_follower_init(); // Inicjalizacja modułu śledzenia linii

  //const uint16_t pwm_r = 980; // Minimalnie niższe wypełnienie PWM dla prawego silnika, by skorygować tor jazdy
  //const uint16_t pwm_l = 999; // Maksymalne wypełnienie PWM (pełna prędkość) dla lewegp silnika

  // Uruchomienie nasłuchiwania na porcie UART1 dla komend Bluetooth
  HAL_UART_Receive_IT(&huart1, (uint8_t*)&uart_rx_buffer, 1);

  HAL_ADC_Start_DMA(&hadc4, (uint32_t*)adc_buffer, ADC_CHANNELS); // Uruchomienie konwersji ADC w trybie DMA
                                                                  // Odczyty z `ADC_CHANNELS` kanałów będą
                                                                  // automatycznie zapisywane do `adc_buffer


  /* USER CODE END 2 */

```

```
// Ustawienie początkowego trybu i stanu
current_robot_mode = ROBOT_MODE_STOPPED; //  Ustawienie początkowego trybu robota na zatrzymany
robot_running = 0; // Domyślnie zatrzymany

// Wysłanie komunikatów powitalnych przez UART
UART_SendString("Robot Gotowy!!!\r\nWyślij 'A' aby włączyć Podążanie po linii, 'M' dla ręcznego sterowania.\r\n");
UART_SendString("Aktualny Tryb: Zatrzymano. Wciśnij PA0 albo wyślij komendę BT.\r\n");

while (1)
{ // sterowanie diodą state (PE8)
    if (HC05_State()) {
        HAL_GPIO_WritePin(GPIOE, GPIO_PIN_8, GPIO_PIN_SET);  // Dioda ON (połączony)
    }
    else {
        HAL_GPIO_WritePin(GPIOE, GPIO_PIN_8, GPIO_PIN_RESET); // Dioda OFF (brak połączenia)
    }
  /* USER CODE END WHILE */

  /* USER CODE BEGIN 3 */
    // switch sterujący robotem w zależności od ustawionego trybu
    switch (current_robot_mode) {
            case ROBOT_MODE_LINE_FOLLOWER:
                if (robot_running) {
                    float distance = HCSR04_ReadDistance(); // Pomiar dystansu

                    if (distance >= 16.0f || distance < 0.0f) { // Jeśli nie ma przeszkody lub błąd czujnika
                        line_follower_process(); // Wywołanie funkcji śledzenia linii
                        HAL_Delay(1); // krótkie opóźnienie
                    } else { // Przeszkoda wykryta
                        robot_stop();
                        UART_SendString("Przeszkoda: Zatrzymano.\r\n");
                        // Opcjonalnie: Zawracanie
                        // UART_SendString("Przeszkoda: Zawracam.\r\n");
                        // HAL_Delay(500);
                        // robot_drive(BT_PWM_SPEED_RIGHT, 1, BT_PWM_SPEED_LEFT, 0);
                        // HAL_Delay(680);
                        // robot_stop();
                        // HAL_Delay(50);
                    }
                } else {
                    robot_stop(); // zatrzymanie po wcisnieciu przycisku
                    line_follower_reset_state();
                    HAL_Delay(200);
                }
                break;

            case ROBOT_MODE_BLUETOOTH_MANUAL:
                // Logika ruchu jest obsługiwana w ProcessBluetoothCommand poprzez przerwanie UART
                HAL_Delay(50); // Krótkie opóźnienie
                break;
```

```
    if (HC05_State()) {
        HAL_GPIO_WritePin(GPIOE, GPIO_PIN_8, GPIO_PIN_SET);  // Dioda ON (połączony)
    }
    else {
        HAL_GPIO_WritePin(GPIOE, GPIO_PIN_8, GPIO_PIN_RESET); // Dioda OFF (brak połączenia)
    }
  /* USER CODE END WHILE */

  /* USER CODE BEGIN 3 */
    // switch sterujący robotem w zależności od ustawionego trybu
    switch (current_robot_mode) {
            case ROBOT_MODE_LINE_FOLLOWER:
                if (robot_running) {
                    float distance = HCSR04_ReadDistance(); // Pomiar dystansu

                    if (distance >= 16.0f || distance < 0.0f) { // Jeśli nie ma przeszkody lub błąd czujnika
                        line_follower_process(); // Wywołanie funkcji śledzenia linii
                        HAL_Delay(1); // krótkie opóźnienie
                    } else { // Przeszkoda wykryta
                        robot_stop();
                        UART_SendString("Przeszkoda: Zatrzymano.\r\n");
                        // Opcjonalnie: Zawracanie
                        // UART_SendString("Przeszkoda: Zawracam.\r\n");
                        // HAL_Delay(500);
                        // robot_drive(BT_PWM_SPEED_RIGHT, 1, BT_PWM_SPEED_LEFT, 0);
                        // HAL_Delay(680);
                        // robot_stop();
                        // HAL_Delay(50);
                    }
                } else {
                    robot_stop(); // zatrzymanie po wcisnieciu przycisku
                    line_follower_reset_state();
                    HAL_Delay(200);
                }
                break;

            case ROBOT_MODE_BLUETOOTH_MANUAL:
                // Logika ruchu jest obsługiwana w ProcessBluetoothCommand poprzez przerwanie UART
                HAL_Delay(50); // Krótkie opóźnienie
                break;

            case ROBOT_MODE_STOPPED:
                // Robot jest zatrzymany, silniki wyłączone
                HAL_Delay(200); // odciążanie CPU
                break;
  }
  /* USER CODE END 3 */
}
}
```

Oliwier Bogdański, Kacper Szponar

Funkcje callback

```c
/* USER CODE BEGIN 4 */
// Funkcja obsługująca przerwanai zewnętrzne wywoływane przez wciśniecie User Button (PA0)
void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)
{
    if (GPIO_Pin == GPIO_PIN_0) // Sprawdza, czy przerwanie pochodzi od przycisku użytkownika (PA0)
    {
        // Prosty debounce
        static uint32_t last_press_time = 0; // Zmienna przechowuje czas ostatniego zarejestrowanego naciśnięcia
        if (HAL_GetTick() - last_press_time < 250) { // 250ms debounce
            return; // Ignoruj to naciśnięcie
        }
        last_press_time = HAL_GetTick();

        if (current_robot_mode == ROBOT_MODE_LINE_FOLLOWER) {
            robot_running = !robot_running; // Przełącz stan start/stop dla line-followera
            if (robot_running) {
                line_follower_reset_state(); // Resetowanie stanu przy starcie
                UART_SendString("PA0: START podążania po linii\r\n");
            } else { // Zatrzymanie robota
                robot_stop();
                UART_SendString("PA0: STOP podążania po linii\r\n");
            }
        } else if (current_robot_mode == ROBOT_MODE_BLUETOOTH_MANUAL) {
            // W trybie manualnym BT, PA0 działa jako nagły stop
            robot_stop();
            UART_SendString("PA0: Zatrzymano ręczne sterowanie\r\n");
        } else if (current_robot_mode == ROBOT_MODE_STOPPED) {
            // Jeśli zatrzymany, PA0 może go przełączyć i uruchomić w trybie Line Follower
            current_robot_mode = ROBOT_MODE_LINE_FOLLOWER;
            robot_running = 1; // Uruchomianie od razu
            line_follower_reset_state();
            UART_SendString("PA0: Zmieniono tryb na podążanie po linii\r\n");
        }
    }
}
```

```
872
873 // funkcja obsługująca przerwania od timera w trybie Input Capture (przechwytywania wejścia)
874 // Wywoływana, gdy timer wykryje zbocze na skonfigurowanym kanale
875 // służy do pomiaru czasu trwania impulsu echo z HC-SR04
876 void HAL_TIM_IC_CaptureCallback(TIM_HandleTypeDef *htim)
877 {
878     // Sprawdzanie, czy przerwanie pochodzi od TIM2 i jego aktywnego kanału 4
879     if (htim->Instance == TIM2 && htim->Channel == HAL_TIM_ACTIVE_CHANNEL_4)
880     {
881         if (echo_captured == 0) // Jeśli to pierwsze (narastające) zbocze impulsu echo
882         {
883             echo_start = HAL_TIM_ReadCapturedValue(htim, TIM_CHANNEL_4); // Odczytaj i zapisz wartość licznika timera (czas startu)
884
885             // Zmień polaryzację przechwytywania na zbocze opadające, aby złapać koniec impulsu
886             __HAL_TIM_SET_CAPTUREPOLARITY(htim, TIM_CHANNEL_4, TIM_INPUTCHANNELPOLARITY_FALLING);
887             echo_captured = 1; // Ustawia flagę, że start echa został przechwycony
888         }
889         else if (echo_captured == 1) // Jeśli to drugie (opadające) zbocze impulsu echo
890         {
891             echo_end = HAL_TIM_ReadCapturedValue(htim, TIM_CHANNEL_4); // Odczytaj i zapisz wartość licznika timera (czas końca)
892
893             // Zmień polaryzację z powrotem na zbocze narastające dla następnego pomiaru
894             __HAL_TIM_SET_CAPTUREPOLARITY(htim, TIM_CHANNEL_4, TIM_INPUTCHANNELPOLARITY_RISING);
895             echo_captured = 2; // Ustaw flagę, że koniec echa został przechwycony (pomiar kompletny)
896         }
897     }
898 }
899
900 // Callback wywoływany po odebraniu danych przez UART (Bluetooth)
901 void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart)
902 {
903   if (huart->Instance == USART1) // Sprawdza, czy przerwanie pochodzi od USART1
904   {
905     ProcessBluetoothCommand(uart_rx_buffer); // Przetwórz odebrany bajt
906
907     // Ponownie włącz nasłuchiwanie na kolejny bajt
908     HAL_UART_Receive_IT(&huart1, (uint8_t*)&uart_rx_buffer, 1);
909
910   }
911 }
```

Oliwier Bogdański, Kacper Szponar

Line_follower.h

plik nagłówkowy dla line_follower zawierający definicje i parametry

```c
/*
 * line_follower.h
 */

#ifndef LINE_FOLLOWER_H_
#define LINE_FOLLOWER_H_

#include "main.h"
#include <stdint.h>

//Konfiguracja Śledzenia Linii
#define ADC_CHANNELS 8 // Liczba czujników linii (kanałów ADC)

#define LINE_FOLLOWER_KP 0.001f // Współczynnik proporcjonalny (P) regulatora PID do korekcji błędu pozycji
#define LINE_FOLLOWER_KD 0.001f // Współczynnik różniczkujący (D) regulatora PID do tłumienia oscylacji

#define LINE_FOLLOWER_BASE_SPEED_PWM 650 // Podstawowa prędkość PWM silników
#define LINE_FOLLOWER_MIN_SPEED_PWM 650 // Minimalna dopuszczalna prędkość PWM silników
#define LINE_FOLLOWER_MAX_SPEED_PWM 850// Maksymalna dopuszczalna prędkość PWM silników

// Definicja bazowych prędkości PWM dla obu silników uwzględniajac kalibrację
#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)
#define RIGHT_MOTOR_BASE_PWM LINE_FOLLOWER_BASE_SPEED_PWM

#define SENSOR_LINE_THRESHOLD 600 // Próg odczytu ADC, powyżej którego uznaje się,
                                  //że czujnik jest nad linią (białe tło, czarna linia)
#define SENSOR_WEIGHT_MULTIPLIER 1000 // Mnożnik używany do obliczania ważonej pozycji linii

// Docelowa wartość pozycji, gdy robot jest idealnie na środku linii
#define LINE_CENTER_POSITION ( ( (ADC_CHANNELS - 1) * SENSOR_WEIGHT_MULTIPLIER ) / 2 )

//Konfiguracja ostrych zakrętów
#define SHARP_TURN_PWM 700 // Prędkość PWM silników podczas wykonywania ostrego zakrętu
#define SHARP_TURN_DURATION_MS 600 // Czas trwania manewru ostrego zakrętu w milisekundach
#define SHARP_TURN_OUTER_SENSORS_ACTIVE 2 // Liczba skrajnych zewnętrznych czujników,
                                          //które muszą wykryć linię, aby zidentyfikować ostry zakręt

#define SHARP_TURN_INNER_SENSORS_INACTIVE 3 // Liczba wewnętrznych czujników (od strony zakrętu),
                                            // które muszą NIE wykrywać linii,
                                            // aby potwierdzić ostry zakręt

//Konfiguracja oszukiwania zgubionej linii
#define LOST_LINE_REVERSE_PWM 710           // Prędkość PWM podczas cofania
#define LOST_LINE_REVERSE_DURATION_MS 400   // Czas cofania po zgubieniu linii
#define LOST_LINE_SWEEP_TURN_PWM 550        // Prędkość PWM podczas obrotu w miejscu przy szukaniu
#define LOST_LINE_SWEEP_DURATION_MS 550     // Czas jednego obrotu przy szukaniu
#define LOST_LINE_MAX_SEARCH_CYCLES 5       // Liczba pełnych cykli poszukiwania (lewo-prawo) zanim robot się podda


// Stany robota
typedef enum {
    STATE_FOLLOWING_LINE,                   // Stan gdy robot jedzie po linii

    // Stany dla zgubionej linii
    STATE_LOST_LINE_INITIATE_SEARCH,        // Stan początkowy po zgubieniu linii
    STATE_LOST_LINE_REVERSING,              // Robot cofa
    STATE_LOST_LINE_SEARCH_LEFT,            // Robot szuka obracając się w lewo w miejscu
    STATE_LOST_LINE_SEARCH_RIGHT,           // Robot szuka obracając się w prawo w miejscu
    STATE_LOST_LINE_FAIL_STOP,              // Robot się poddaje i zatrzymuje

    // Stany dla ostrych zakrętów
    STATE_DETECTED_SHARP_LEFT_TURN,
    STATE_PERFORMING_SHARP_LEFT_TURN,
    STATE_DETECTED_SHARP_RIGHT_TURN,
    STATE_PERFORMING_SHARP_RIGHT_TURN,
} RobotLineState;

extern uint16_t adc_buffer[ADC_CHANNELS]; // Bufor przechowujący odczyty z czujników linii (ADC)

void line_follower_init(void);            // Funkcja inicjalizująca moduł śledzenia linii
void line_follower_process(void);         // Główna funkcja przetwarzająca logikę śledzenia linii, wywoływana cyklicznie
void line_follower_reset_state(void);     // Funkcja resetująca stan robota do początkowego (np. STATE_FOLLOWING_LINE)

#endif /* LINE_FOLLOWER_H_ */
```

Oliwier Bogdański, Kacper Szponar

Line_follower.c

Zmienne

```c
#include "line_follower.h" //import zmiennych i definicji z pliku nagłówkowego
#include "main.h"
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

// Tablica wag przypisanych do każdego czujnika. Wagi rosną od lewej do prawej,
// co pozwala na obliczenie środka ciężkości wykrytej linii.
static const int32_t sensor_weights[ADC_CHANNELS] = {
    0 * SENSOR_WEIGHT_MULTIPLIER, 1 * SENSOR_WEIGHT_MULTIPLIER, 2 * SENSOR_WEIGHT_MULTIPLIER, 3 * SENSOR_WEIGHT_MULTIPLIER,
    4 * SENSOR_WEIGHT_MULTIPLIER, 5 * SENSOR_WEIGHT_MULTIPLIER, 6 * SENSOR_WEIGHT_MULTIPLIER, 7 * SENSOR_WEIGHT_MULTIPLIER
};

// Statyczne zmienne modułu przechowujące jego wewnętrzny stan

static float line_last_error_for_decision = 0.0f; // Ostatni zarejestrowany błąd pozycji linii,
//używany do podjęcia decyzji o kierunku poszukiwania linii po jej zgubieniu.
//Aktualizowany tylko, gdy linia jest wykrywana stabilnie

static float line_last_error_for_pid = 0.0f;      // Ostatni zarejestrowany błąd pozycji linii,
//używany do obliczenia członu różniczkującego regulatora PID

static int32_t last_known_position = LINE_CENTER_POSITION; // Ostatnia znana pozycja linii, nawet jeśli była chwilowa
static RobotLineState_t current_robot_state = STATE_FOLLOWING_LINE; // Aktualny stan robota
static uint32_t state_timer = 0; // Zmienna używana do odmierzania czasu trwania określonych stanów lub operacji
static uint8_t search_cycle_counter = 0; // Licznik cykli poszukiwania wykonanych podczas szukania zgubionej linii

```

Oliwier Bogdański, Kacper Szponar

Funkcje pomocnicze

```c
// Prototypy funkcji prywatnych
static int32_t calculate_line_position(uint16_t *sensor_values);
static void handle_lost_line(void);
static void handle_sharp_turns(int32_t position, uint16_t *sensor_values);


// Funkcja pomocnicza inicjalizująca stan modułu szukania linii. Wytołana tylko raz

void line_follower_init(void) {
    line_last_error_for_decision = 0.0f;
    line_last_error_for_pid = 0.0f;
    last_known_position = LINE_CENTER_POSITION; // Zakładamy, że na starcie robot jest na środku linii
    current_robot_state = STATE_FOLLOWING_LINE; // Domyślny stan początkowy.
    state_timer = HAL_GetTick(); // Inicjalizacja prostego timera stanów za pomocą HAL
    search_cycle_counter = 0; // Zmienna zliczająca ilość cykli poszukiwania
}

// Resetuje stan modułu, pozwalając na uruchomienie od npwa podążania za linią
void line_follower_reset_state(void) {
    line_last_error_for_decision = 0.0f;
    line_last_error_for_pid = 0.0f;
    current_robot_state = STATE_FOLLOWING_LINE;
    search_cycle_counter = 0;
}

// Funkcja pomocnicza obliczająca pozycję linii na podstawie odczytów z czujników odbiciowych
// Zwraca wartość pozycji lub -1, jeśli linia nie została wykryta.

static int32_t calculate_line_position(uint16_t *sensor_values) {
    uint32_t weighted_sum = 0;          // Suma ważona (wartość czujnika * waga czujnika)
    uint16_t sum_of_active_sensors = 0; // Suma wartości aktywnych czujników
    uint8_t active_sensor_count = 0;    // Liczba czujników, które wykryły linie
    uint8_t first_active_sensor = ADC_CHANNELS; // Indeks pierwszego aktywnego czujnika od lewej

    // Czujnik jest uznawany za aktywny, jeśli jego odczyt przekracza próg SENSOR_LINE_THRESHOLD
    for (uint8_t i = 0; i < ADC_CHANNELS; i++) {
        if (sensor_values[i] > SENSOR_LINE_THRESHOLD) {
            weighted_sum += (uint32_t)sensor_values[i] * sensor_weights[i];
            sum_of_active_sensors += sensor_values[i];
            active_sensor_count++;
            if (i < first_active_sensor) { // Zapamiętanie indeksu pierwszego aktywnego czujnika.
                first_active_sensor = i;
            }
        }
    }

    if (active_sensor_count == 0) {
        return -1; // -1 oznacza że, linia nie została wykryta przez żaden czujnik
    }

    // Jeśli wszystkie czujniki widzą linię (np szeroka linia) to uznajemy, że robot jest na środku
    if (active_sensor_count == ADC_CHANNELS) {
        last_known_position = LINE_CENTER_POSITION;
        return LINE_CENTER_POSITION;
    }

    // Jeśli tylko jeden czujnik jest aktywny, jego pozycja jest traktowana jako pozycja linii.
    if (active_sensor_count == 1) {
        if (first_active_sensor == 0) { // Aktywny tylko skrajny lewy czujnik.
            last_known_position = sensor_weights[0];
            return sensor_weights[0];
        }
        if (first_active_sensor == ADC_CHANNELS - 1) { // Aktywny tylko skrajny prawy czujnik.
            last_known_position = sensor_weights[ADC_CHANNELS - 1];
            return sensor_weights[ADC_CHANNELS - 1];
        }
    }
    // Jeśli aktywny jest jeden ze środkowych czujników to obliczana jest średnia ważona
    // Normalizacja wyniku daje większą wagę czujnikom z silniejszym sygnałem
    last_known_position = weighted_sum / sum_of_active_sensors;
    return last_known_position; // Zwracamy pozycję
}
```

Główna logika

```c
// Główna funkcja przetwarzająca logikę podążania za linią wywoływana cyklicznie

void line_follower_process(void) {
    // Obliczanie aktualnej pozycji linii na podstawie danych z bufora ADC
    int32_t current_position = calculate_line_position(adc_buffer);

    //Debug UART do sprawdzania wartości wskazywanych przez czujniki
    char uart_buf[120];
    sprintf(uart_buf, "ADC:[%4u %4u %4u %4u %4u %4u %4u %4u] Pos:%ld State:%d\r\n",
            adc_buffer[0], adc_buffer[1], adc_buffer[2], adc_buffer[3],
            adc_buffer[4], adc_buffer[5], adc_buffer[6], adc_buffer[7],
            current_position, current_robot_state);
     UART_SendString(uart_buf);

    // W pierwszej kolejności sprawdzamy czy nie ma zakrętu
    handle_sharp_turns(current_position, adc_buffer);
    // Jeśli robot jest w trakcie zakrętu pomijamy resztę logiki
    if (current_robot_state == STATE_PERFORMING_SHARP_LEFT_TURN ||
        current_robot_state == STATE_PERFORMING_SHARP_RIGHT_TURN) {
        return;
    }

    // Linia jest znaleziona
    if (current_position != -1) {

        current_robot_state = STATE_FOLLOWING_LINE; // Ustawienie stanu robota na podążanie za linią
        search_cycle_counter = 0; // Zresetowanie licznika cykli poszukiwania bo linia została znaleziona

        // Obliczenie błędu jako odchylenia aktualnej pozycji robota od idealnej pozycji linii
        float error = (float)current_position - LINE_CENTER_POSITION;

        // Aktualizacja zmiennej line_last_error_for_decision tylko wtedy,
        //gdy błąd nie jest ekstremalny czyli np kiedy linia nie jest na skraju
        uint8_t active_sensors_count_temp = 0;
        for(int i=0; i<ADC_CHANNELS; ++i) if(adc_buffer[i] > SENSOR_LINE_THRESHOLD) active_sensors_count_temp++;

        if (abs((int)error) < (LINE_CENTER_POSITION * 0.85f) && active_sensors_count_temp > 0) {
            line_last_error_for_decision = error;
        }

        // Obliczenie członu różniczkującego (D) regulatora PID.
        float derivative = error - line_last_error_for_pid;
        // Obliczenie korekty prędkości silników za pomocą regulatora PD (Proporcjonalno-Różniczkującego)
        float motor_speed_correction = (LINE_FOLLOWER_KP * error) + (LINE_FOLLOWER_KD * derivative);
        // Zapisanie bieżącego błędu do użycia w następnej iteracji jako `line_last_error_for_pid`
        line_last_error_for_pid = error;
```

```
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
```

Funkcja obsługująca zgubienie linii

```c
// Funkcja obsługująca logikę zachowania robota po zgubieniu linii
// Linia jest poszukiwana sekwencyjnie do momentu aż robot znajdzie linię w ciągu ustalonej liczby cykli poszukiwania
static void handle_lost_line(void) {
    uint32_t current_time = HAL_GetTick(); // Pobranie aktualnego czasu systemowego.

    switch (current_robot_state) {
    // Inicjalizacja sekwencji
        case STATE_LOST_LINE_INITIATE_SEARCH:
            // Najpierw zaczyna on cofnięcia
            current_robot_state = STATE_LOST_LINE_REVERSING;
            state_timer = current_time; // Zapisz czas rozpoczęcia cofania.
            search_cycle_counter = 0;   // Zresetuj licznik cykli dla nowej sekwencji poszukiwania
            robot_drive(LOST_LINE_REVERSE_PWM, 0, LOST_LINE_REVERSE_PWM, 0); // Cofanie prosto z ustalonym PWM dla cofania
            break;

        case STATE_LOST_LINE_REVERSING:
            // Sprawdzanie czy nie upłynął czas przeznaczony na cofanie
            if (current_time - state_timer > LOST_LINE_REVERSE_DURATION_MS) {
                // Po cofnięciu robot decyduje o pierwszym kierunku obrotu w miejscu
                // Decyzja jest podjęta na podstawie wartości `line_last_error_for_decision` lub, jeśli jest on bliski zera, na `last_known_position`
                // Jeśli linia była po prawej (błąd > 0 lub LKP > centrum), to zaczyna szukać w prawo

                // LKP - last_known_position

                if (line_last_error_for_decision > SENSOR_WEIGHT_MULTIPLIER / 4 || (abs((int)line_last_error_for_decision) <=
                        SENSOR_WEIGHT_MULTIPLIER / 4 && last_known_position > LINE_CENTER_POSITION)) {
                    current_robot_state = STATE_LOST_LINE_SEARCH_RIGHT;
                    robot_drive(LOST_LINE_SWEEP_TURN_PWM, 0, LOST_LINE_SWEEP_TURN_PWM, 1); // Obrót w prawo
                } else { // Linia była po lewej, na środku, lub błąd był nieznaczny i LKP <= centrum
                    current_robot_state = STATE_LOST_LINE_SEARCH_LEFT;
                    robot_drive(LOST_LINE_SWEEP_TURN_PWM, 1, LOST_LINE_SWEEP_TURN_PWM, 0); // Obrót w lewo
                }
                state_timer = current_time; // Zapisz czas rozpoczęcia obrotu
            } else {
                // Kontynuuj cofanie, jeśli czas nie upłynął
                robot_drive(LOST_LINE_REVERSE_PWM, 0, LOST_LINE_REVERSE_PWM, 0);
            }
            break;
```

```
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 }
```

Oliwier Bogdański, Kacper Szponar

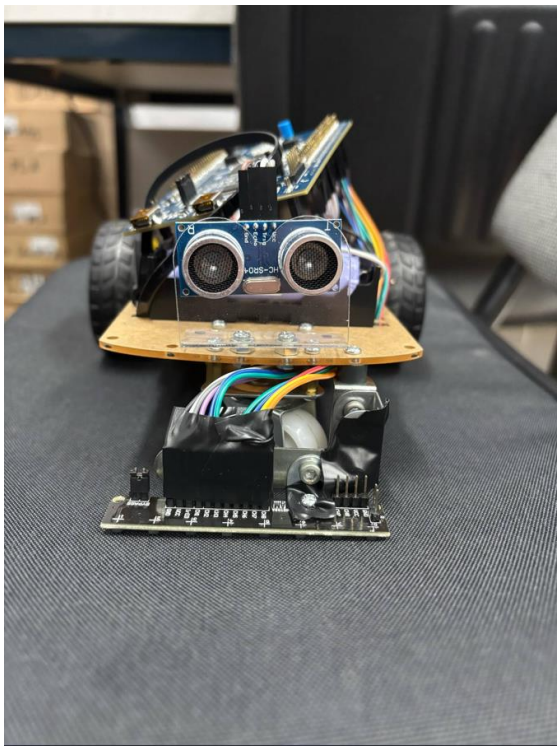
Zakręty

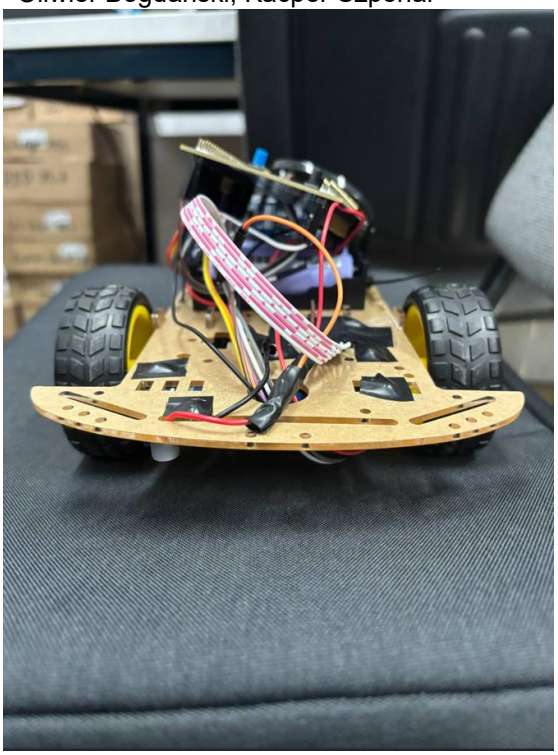
```
// Funkcja obsługująca wykrywanie i wykonywanie ostrych zakrętów.
static void handle_sharp_turns(int32_t position, uint16_t *sensor_values) {
    uint8_t active_left_outer = 0;  // Liczba aktywnych czujników na skrajnej lewej stronie
    uint8_t active_right_outer = 0; // Liczba aktywnych czujników na skrajnej prawej stronie
    uint8_t active_center = 0;      // Liczba aktywnych czujników w środkowej części
    uint8_t total_active = 0;       // Całkowita liczba aktywnych czujników

    // Zliczanie aktywnych czujników w poszczególnych strefach
    // Zmienna SHARP_TURN_OUTER_SENSORS_ACTIVE definiuje, ile skrajnych czujników jest branych pod uwagę
    for(int i = 0; i < ADC_CHANNELS; i++) {
        if (sensor_values[i] > SENSOR_LINE_THRESHOLD) {
            total_active++;
            if (i < SHARP_TURN_OUTER_SENSORS_ACTIVE) active_left_outer++;
            else if (i >= ADC_CHANNELS - SHARP_TURN_OUTER_SENSORS_ACTIVE) active_right_outer++;
            else active_center++;
        }
    }

    uint32_t current_time = HAL_GetTick(); // Pobranie czasu

    // Logika wykonywania ostrego zakrętu w lewo
    if (current_robot_state == STATE_PERFORMING_SHARP_LEFT_TURN) {
        // Zakończenie manewru skrętu, jeśli upłynął zdefiniowany czas lub jeśli środkowe czujniki ponownie wykryły linię
        if (current_time - state_timer > SHARP_TURN_DURATION_MS || (active_center > 0 && total_active < ADC_CHANNELS -1 && total_active > 0)) {
            int32_t pos_after_turn = calculate_line_position(sensor_values); // Sprawdzamy pozycję lini po zakręcie
            // Jeśli linia jest znaleziona i widoczna przez środkowe czujniki to ustawiamy odpowiedni stan
            if (pos_after_turn != -1 && active_center > 0) {
                current_robot_state = STATE_FOLLOWING_LINE;
                line_last_error_for_pid = 0; // Reset błędu PID w celu uniknięcia gwałtownej reakcji
                robot_drive(LINE_FOLLOWER_BASE_SPEED_PWM, 1, LINE_FOLLOWER_BASE_SPEED_PWM, 1);
            } else { // Linia nie została znaleziona lub nie jest widoczna centralnie po zakręcie
                current_robot_state = STATE_LOST_LINE_INITIATE_SEARCH; // Rozpocznij poszukiwanie linii
                // handle_lost_line zostanie wywołane w głównej pętli line_follower_process
            }
        } else {
             // Kontynuacja wykonywania ostrego skrętu w lewo
             robot_drive(SHARP_TURN_PWM, 1, (uint16_t)(SHARP_TURN_PWM * 0.3f), 0);
        }
        return; // Zakończenie jeśli wykonujemy zakręt
    }

    // Logika wykonywania ostrego zakrętu w prawo podobnie jak w lewo
    if (current_robot_state == STATE_PERFORMING_SHARP_RIGHT_TURN) {
        if (current_time - state_timer > SHARP_TURN_DURATION_MS || (active_center > 0 && total_active < ADC_CHANNELS-1 && total_active > 0 )) {
            int32_t pos_after_turn = calculate_line_position(sensor_values);
            if (pos_after_turn != -1 && active_center > 0) {
                current_robot_state = STATE_FOLLOWING_LINE;
                line_last_error_for_pid = 0;
                robot_drive(LINE_FOLLOWER_BASE_SPEED_PWM, 1, LINE_FOLLOWER_BASE_SPEED_PWM, 1);
            } else {
                current_robot_state = STATE_LOST_LINE_INITIATE_SEARCH;
```

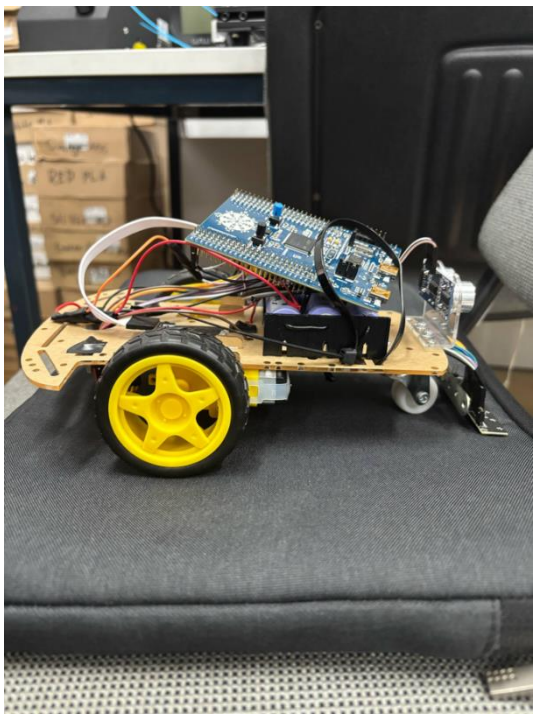
```
331                 current_robot_state = STATE_FOLLOWING_LINE;
332                 line_last_error_for_pid = 0;
333                 robot_drive(LINE_FOLLOWER_BASE_SPEED_PWM, 1, LINE_FOLLOWER_BASE_SPEED_PWM, 1);
334             } else {
335                 current_robot_state = STATE_LOST_LINE_INITIATE_SEARCH;
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 
```

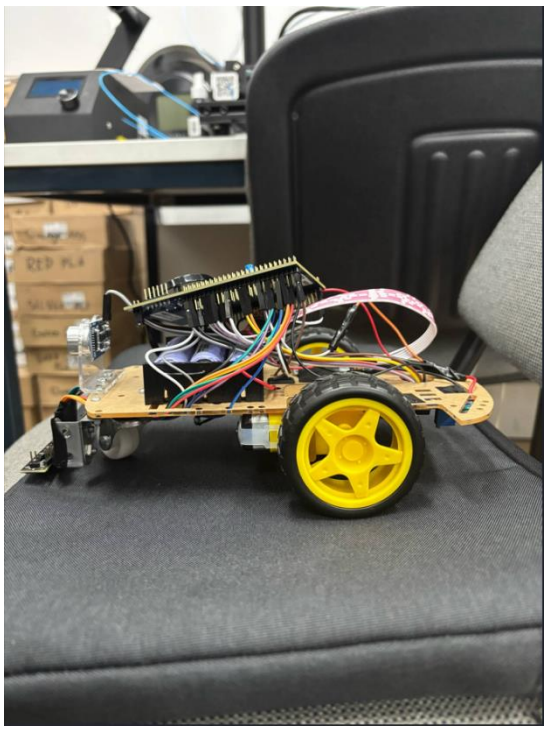
6. Zdjęcia opracowanego robota

Oliwier Bogdański, Kacper Szponar









Oliwier Bogdański, Kacper Szponar