

Raport z budowy robota

Adrian Popielarczyk 21295, Zuzanna Orzechowska 21284

Opis robota

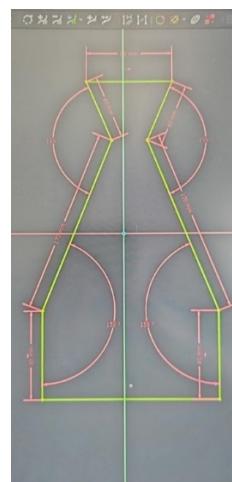
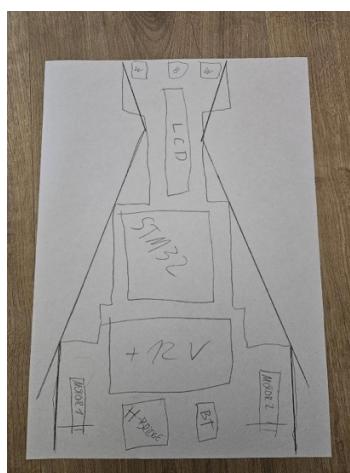
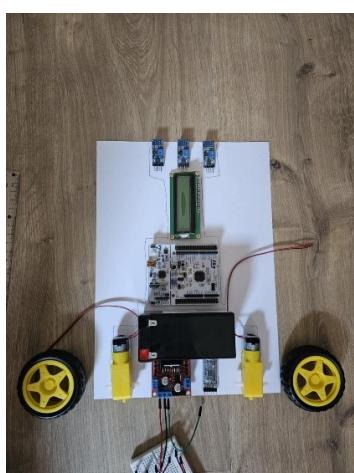
Robot potrafi poruszać się po dowolnym czarnym śladzie na białym tle (linii) z satysfakcjonującą płynnością. Robot posiada ekran LCD, potrafiący wyświetlać parametry oraz napisy. Realizuje te zadanie za pomocą zewnętrznych przycisków. Robot posiada 1 parę osi. Oś nie jest skrętna, a przyczepność oraz napęd występuje na tylnej osi. Skrót przypomina schemat poruszania się pojazdami pełzającymi jak czołg, z tą różnicą, że zamiast gąsienic mamy do czynienia z kołami. Robot posiada moduł Bluetooth umożliwiający zdalne sterowanie. Robot posiada jedno źródło zasilania – do silników, oraz do płytki STM 32.

Elementy wybrane do budowy robota

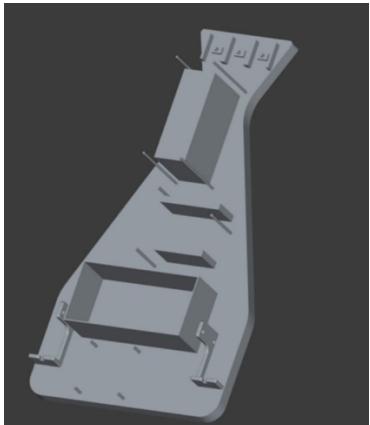
- 1x Płyta STM32 Nucleo L010RB
- 2x Silnik z przekładnią 48:1, o zakresie pracy 3-6V
- 1x Mostek typu H, model L298n
- 1x Akumulator żelowy 12V, 1.3Ah
- 1x Przełącznik On/Off
- Kable do łączeń
- Połowa korka po winie
- Kostka wydrukowana w drukarce 3D

Mechanika robota

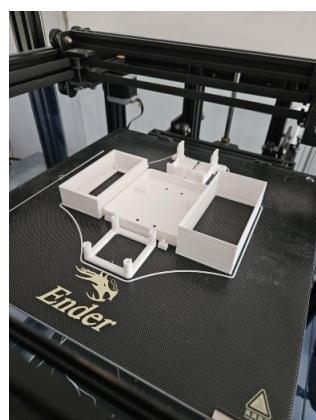
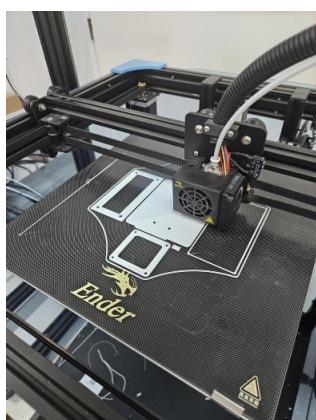
Model koncepcyjny wraz z projektem i budową podwozia, projekt graficzny został wykonany w programie Freecad:



Adrian Popielarczyk 21295, Zuzanna Orzechowska 21284



Elementy trzymające wydrukowane zostały na drukarce 3D – Ender 6, poniżej pokazany jest ich druk:



Elementy poza podkładkami pod sensory odbicia, zostały wzięte z poniższych źródeł, autorzy elementów wskazani są również na tych samych stronach:

<https://www.printables.com/model/59983-16x2-lcd-with-d1-mini-case/files>

<https://www.thingiverse.com/thing:6121113/files>

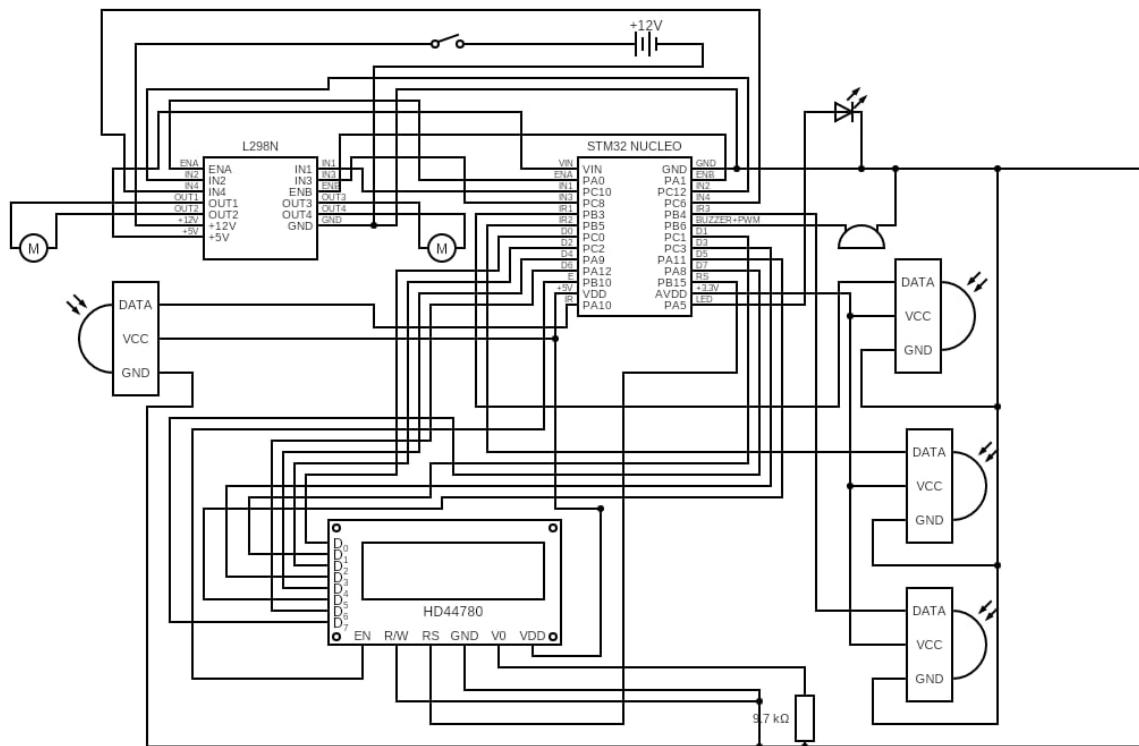
<https://www.printables.com/model/161725-din-rail-case-for-stm32-nucleo-64-boards/files>

<https://www.printables.com/model/552244-sparky-tt-motor-mounts>

Schemat elektroniczny robota na kolejnej stronie →

Adrian Popielarczyk 21295, Zuzanna Orzechowska 21284

Schemat elektryczny robota



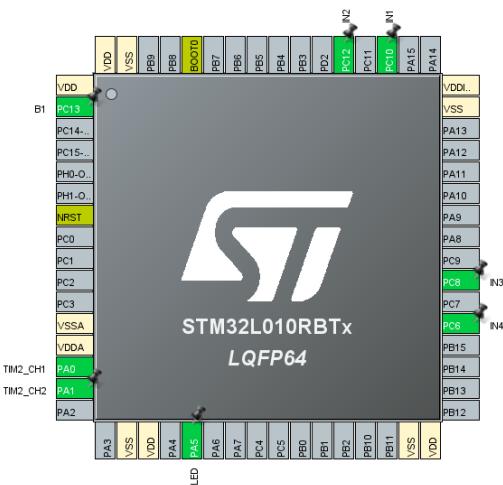
Na kolejnej stronie przedstawiono postępy dla Milestone I →

Adrian Popielarczyk 21295, Zuzanna Orzechowska 21284

Milestone I – 13.05.2025

Oprogramowanie sterujące

Konfiguracja pinów:



Wyjścia pinów podłączone do odpowiadających deskrypcją pinów na mostku H.

Płytkę działa z zegarem 16 MHz, prescaler timerów ustawiono na 0, a maksymalne wypełnienie na 1600. Uzyskano dzięki temu częstotliwość PWM ok. 10 000Hz.

Kod programu głównego, wykonujący 10 sekwencji po kolej:

Kod w sekcji USER CODE WHILE:

```
while (1)
{
    while(HAL_GPIO_ReadPin(GPIOC, B1_Pin) == GPIO_PIN_SET) //następuje odczytanie stanu przycisku na
    porcie C, jeśli on zostanie
    //wciśnięty i zaczyna rozpoczęwanie sekwencji
    {
        HAL_GPIO_TogglePin(GPIOA, LED_Pin); // po uruchomieniu – włączeniu przycisku dioda LED zaczyna
        świecić z opóźnieniem 200ms
        HAL_Delay(200);
    }

    //następuje reset wszystkich pinów dla kierunków kanału mostka H na porcie C – początkowo motory
    sterujące ruchem kół zostają wyłączone
    //stąd stan GPIO_PIN_RESET
    HAL_GPIO_WritePin(GPIOC, IN1_Pin, GPIO_PIN_RESET);
    HAL_GPIO_WritePin(GPIOC, IN2_Pin, GPIO_PIN_RESET);
    HAL_GPIO_WritePin(GPIOC, IN3_Pin, GPIO_PIN_RESET);
    HAL_GPIO_WritePin(GPIOC, IN4_Pin, GPIO_PIN_RESET);
}
```



Raport z budowy robota

Adrian Popielarczyk 21295, Zuzanna Orzechowska 21284

```
//następuje inicjalizacja PWM na dwóch kanałach Timera – pierwszym i drugim, dla dwóch motorów
HAL_TIM_PWM_Start(&htim2, TIM_CHANNEL_1);
HAL_TIM_PWM_Start(&htim2, TIM_CHANNEL_2);

// po inicjalizacji początkowe wartości dla obu kanałów ustawione zostały na 0,
// timer i PWM są aktywowane, ale początkowo koła nie dostają żadnych sygnałów, tak zwany startowy
stan bezpieczny
__HAL_TIM_SET_COMPARE(&htim2, TIM_CHANNEL_1, 0);
__HAL_TIM_SET_COMPARE(&htim2, TIM_CHANNEL_2, 0);

//po udanym odpaleniu zaświeca się dioda Led na pinie PA5 – wskazuje na gotowość działania robota i
poruszania się
HAL_GPIO_WritePin(LED_GPIO_Port, LED_Pin, GPIO_PIN_SET);

uint32_t duty = 1200; //aby koła mogły się poruszać ustawiono sygnał dla PWM na 1200 – wskazuje jak
szybko mają się poruszać koła, w tym przypadku będą się one poruszały z 75% prędkości maksymalnego
wypełnienia = 1600
for (int i = 0; i < 10; ++i) // pętla przechodzi przez wszystkie 10 sekwencji
{
switch(i)
{
//ponizej znajdują się kody wykonujące 10 rozkazów sekwencji ruchu pojazdu
case 0: // pierwsza sekwencja – jazda do przodu, IN1 i IN2 są odpowiedzialne za lewy motor, IN3 i
IN4 są odpowiedzialne za prawy motor
HAL_GPIO_WritePin(GPIOC, IN1_Pin, GPIO_PIN_SET); //IN 1 ma wartość 1
HAL_GPIO_WritePin(GPIOC, IN2_Pin, GPIO_PIN_RESET); // IN2 ma wartość 0, zgodnie z zasadą działania
mostka H
//taka konfiguracja wskazuje ze sygnał PWM "idzie" w jednym kierunku, co sprawia, że motor obraca
się do przodu
HAL_GPIO_WritePin(GPIOC, IN3_Pin, GPIO_PIN_SET); // taka sama konfiguracja występuje dla prawego
motoru
HAL_GPIO_WritePin(GPIOC, IN4_Pin, GPIO_PIN_RESET);
__HAL_TIM_SET_COMPARE(&htim2, TIM_CHANNEL_1, duty); //ustawienie wypełnienia sygnału PWM na 1200 –
koła nabierają 75% maksymalnej prędkości
__HAL_TIM_SET_COMPARE(&htim2, TIM_CHANNEL_2, duty); // to samo dzieje się dla drugiego motora pod-
łączonego do kanału 2
HAL_Delay(2500); //taka pojedyncza sekwencja trwa 2,5 sekundy
break;
case 1: //druga sekwencja – jazda do tyłu – wartości są odwrotne co do tych z jazdy do przodu,
czyli
HAL_GPIO_WritePin(GPIOC, IN1_Pin, GPIO_PIN_RESET); // wartość 0 oraz 1 sprawia, że sygnał PWM
"idzie" w drugim kierunku kanału – następuje obrót kół do tyłu
HAL_GPIO_WritePin(GPIOC, IN2_Pin, GPIO_PIN_SET);
HAL_GPIO_WritePin(GPIOC, IN3_Pin, GPIO_PIN_RESET); // to samo ustawienie dla drugiego motoru
HAL_GPIO_WritePin(GPIOC, IN4_Pin, GPIO_PIN_SET);
__HAL_TIM_SET_COMPARE(&htim2, TIM_CHANNEL_1, duty); //wypełnienie pozostaje to samo – 1200
__HAL_TIM_SET_COMPARE(&htim2, TIM_CHANNEL_2, duty);
HAL_Delay(2500);
break;
case 2: //trzecia sekwencja – skręt w lewo, tylko jedne pin IN2 od lewego motoru jest ustawiony na
1
HAL_GPIO_WritePin(GPIOC, IN1_Pin, GPIO_PIN_RESET);
HAL_GPIO_WritePin(GPIOC, IN2_Pin, GPIO_PIN_SET); //sterowanie drugim tranzystorem dla lewego mo-
toru, silnik nie porusza się, ale obraca w odpowiednią stronę
HAL_GPIO_WritePin(GPIOC, IN3_Pin, GPIO_PIN_RESET); //prawy motor nie dostaje żadnych sygnałów
HAL_GPIO_WritePin(GPIOC, IN4_Pin, GPIO_PIN_RESET);
__HAL_TIM_SET_COMPARE(&htim2, TIM_CHANNEL_1, 0); //lewymotor nie dostaje sygnału PWM który spra-
wiłby, że ten zacznie się obracać
__HAL_TIM_SET_COMPARE(&htim2, TIM_CHANNEL_2, duty);
HAL_Delay(1000); // sekwencja trwa 1 sekundę
break;
case 3: // czwarta sekwencja – skręt w prawo – działa to na podobnej zasadzie co skręt w lewo
```



Raport z budowy robota

Adrian Popielarczyk 21295, Zuzanna Orzechowska 21284

```
HAL_GPIO_WritePin(GPIOC, IN1_Pin, GPIO_PIN_SET);
HAL_GPIO_WritePin(GPIOC, IN2_Pin, GPIO_PIN_RESET);
HAL_GPIO_WritePin(GPIOC, IN3_Pin, GPIO_PIN_RESET);
HAL_GPIO_WritePin(GPIOC, IN4_Pin, GPIO_PIN_RESET);
__HAL_TIM_SET_COMPARE(&htim2, TIM_CHANNEL_1, duty);
__HAL_TIM_SET_COMPARE(&htim2, TIM_CHANNEL_2, 0);
HAL_Delay(1000);
break;
case 4: // sekwencja HARD STOP - robot zatrzymuje się, wszystkie piny otrzymują sygnał 0
HAL_GPIO_WritePin(GPIOC, IN1_Pin, GPIO_PIN_RESET);
HAL_GPIO_WritePin(GPIOC, IN2_Pin, GPIO_PIN_RESET);
HAL_GPIO_WritePin(GPIOC, IN3_Pin, GPIO_PIN_RESET);
HAL_GPIO_WritePin(GPIOC, IN4_Pin, GPIO_PIN_RESET);
__HAL_TIM_SET_COMPARE(&htim2, TIM_CHANNEL_1, duty);
__HAL_TIM_SET_COMPARE(&htim2, TIM_CHANNEL_2, duty);

HAL_Delay(3000); // zatrzymanie trwa 3 sekundy
break;
case 5: // sekwencja - lewy silnik do przodu, prawy do tyłu - obrót w miejscu (lewo)
HAL_GPIO_WritePin(GPIOC, IN1_Pin, GPIO_PIN_SET); // lewy silnik porusza się do przodu, z racji iż
sygnał z PWM przebiega w jednym kierunku
HAL_GPIO_WritePin(GPIOC, IN2_Pin, GPIO_PIN_RESET);
HAL_GPIO_WritePin(GPIOC, IN3_Pin, GPIO_PIN_RESET); // prawy silnik obraca się do tyłu, sygnał z PWM
przebiega w odwrotnym kierunku
HAL_GPIO_WritePin(GPIOC, IN4_Pin, GPIO_PIN_SET);
__HAL_TIM_SET_COMPARE(&htim2, TIM_CHANNEL_1, duty);
__HAL_TIM_SET_COMPARE(&htim2, TIM_CHANNEL_2, duty);
HAL_Delay(2500);
break;
case 6: // sekwencja lewy silnik do tyłu, prawy do przodu - obrót w miejscu (prawo)
HAL_GPIO_WritePin(GPIOC, IN1_Pin, GPIO_PIN_RESET);
HAL_GPIO_WritePin(GPIOC, IN2_Pin, GPIO_PIN_SET);
HAL_GPIO_WritePin(GPIOC, IN3_Pin, GPIO_PIN_SET);
HAL_GPIO_WritePin(GPIOC, IN4_Pin, GPIO_PIN_RESET);
__HAL_TIM_SET_COMPARE(&htim2, TIM_CHANNEL_1, duty);
__HAL_TIM_SET_COMPARE(&htim2, TIM_CHANNEL_2, duty);
HAL_Delay(2500);
break;
case 7: // sekwencja jazdy do przodu wolniej
HAL_GPIO_WritePin(GPIOC, IN1_Pin, GPIO_PIN_SET);
HAL_GPIO_WritePin(GPIOC, IN2_Pin, GPIO_PIN_RESET);
HAL_GPIO_WritePin(GPIOC, IN3_Pin, GPIO_PIN_SET);
HAL_GPIO_WritePin(GPIOC, IN4_Pin, GPIO_PIN_RESET);
__HAL_TIM_SET_COMPARE(&htim2, TIM_CHANNEL_1, duty / 2); // prędkość jest dwukrotnie ograniczona,
stąd motory poruszają się dwa razy wolniej
__HAL_TIM_SET_COMPARE(&htim2, TIM_CHANNEL_2, duty / 2);
HAL_Delay(4000); // sekwencja trwa 4 sekundy
break;
case 8: // sekwencja jazdy do przodu szybciej
HAL_GPIO_WritePin(GPIOC, IN1_Pin, GPIO_PIN_SET);
HAL_GPIO_WritePin(GPIOC, IN2_Pin, GPIO_PIN_RESET);
HAL_GPIO_WritePin(GPIOC, IN3_Pin, GPIO_PIN_SET);
HAL_GPIO_WritePin(GPIOC, IN4_Pin, GPIO_PIN_RESET);
__HAL_TIM_SET_COMPARE(&htim2, TIM_CHANNEL_1, duty*1.20); // prędkość jest zwiększena do ok 95% maksymalnego wypełnienia PWM
__HAL_TIM_SET_COMPARE(&htim2, TIM_CHANNEL_2, duty*1.20);
HAL_Delay(1000); // sekwencja trwa 1 sekunde
break;
case 9: // sekwencja łagodnego SOFT STOP z mrugnięciem LEDem
HAL_GPIO_WritePin(GPIOC, IN1_Pin, GPIO_PIN_SET); // wszystkie piny dla obu motorów otrzymują sygnał
1
HAL_GPIO_WritePin(GPIOC, IN2_Pin, GPIO_PIN_SET);
HAL_GPIO_WritePin(GPIOC, IN3_Pin, GPIO_PIN_SET);
```

Adrian Popielarczyk 21295, Zuzanna Orzechowska 21284

```
HAL_GPIO_WritePin(GPIOC, IN4_Pin, GPIO_PIN_SET);
__HAL_TIM_SET_COMPARE(&htim2, TIM_CHANNEL_1, 0); // oba timery nie dostają żadnego sygnału - PWM
ustawiony na 0
__HAL_TIM_SET_COMPARE(&htim2, TIM_CHANNEL_2, 0);
HAL_GPIO_WritePin(LED_GPIO_Port, LED_Pin, GPIO_PIN_SET); // Led zapala się i po 0.2 sekundach zga-
szą się
HAL_Delay(200);
HAL_GPIO_WritePin(LED_GPIO_Port, LED_Pin, GPIO_PIN_RESET); // po mrugnięciu LED zostaje wyłączony
break;
}

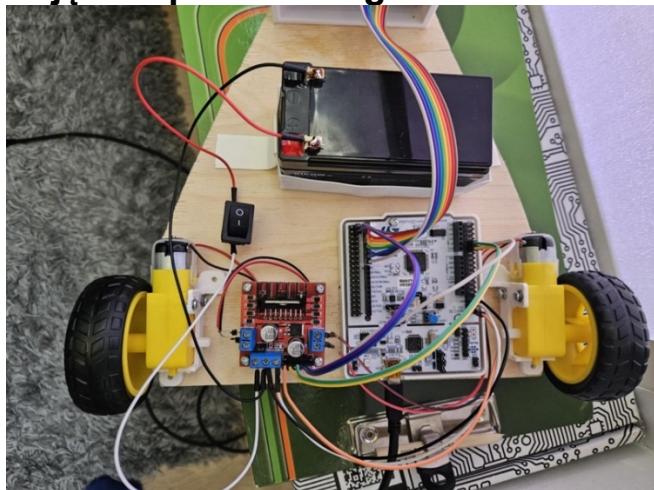
}

/* USER CODE END WHILE */

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

Wnioski: Robot bez problemu wykonuje wszystkie 10 sekwencji po podłączeniu prostownika do akumulatora żelowego. Akumulator żelowy użyty do zasilania jest najprawdopodobniej mocno zużyty gdyż dopiero po podłączeniu go pod prostownik z napięciem 13,5V i korzystaniu z napięcia odpowiadającemu jego znamionowej, umożliwia to ruch robota bez problemu. Próbowano zastosować dodatkowe osobne zasilanie dla płytki STM32, jednakże nie przyniosło to oczekiwanych rezultatów, w związku z czym potrzebny będzie inny akumulator żelowy, o większym napięciu, najlepiej w granicach 14V. Zaprogramowane sekwencje będą mogły zostać użyte do dalszych prac nad robotem.

Zdjęcia opracowanego robota



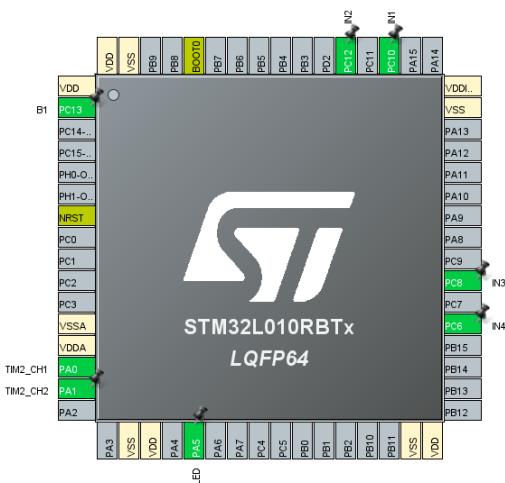


Adrian Popielarczyk 21295, Zuzanna Orzechowska 21284

Milestone II – 27.05.2025

Oprogramowanie sterujące

Konfiguracja pinów:



Poniżej znajduje się opis fragmentów głównego programu - main.c zawierającego kod pozwalający na sterowanie i ruch silników robota

Kod w sekcji USER CODE Includes:

```
/* USER CODE BEGIN Includes */
#include <string.h>
#include "stm32l0xx_hal.h" //biblioteka HAL dla płytka STM32L0
#include <lcd_hd44780.h> //implementacja własnej biblioteki obsługującej wyświetlacz LCD HD44780
/* USER CODE END Includes */
```

Kod w sekcji USER CODE PV:

```
/* USER CODE BEGIN PV */
int lewy, prawy, srodek; // zmienne do odczytu stanu czujników linii, lewy to czujnik na lewo, prawy to czujnik na prawo, srodek to czujnik na środku
uint32_t duty = 1400; // zmienna do przechowywania wartości wypełnienia PWM, 1400 to wartość dla pełnej prędkości
int searchDir = 0; // zmienna do przechowywania kierunku poszukiwania linii, 0 to brak poszukiwania, 1 to szukanie w lewo, -1 to szukanie w prawo
#define DUTY_FAST 1400 // wartość wypełnienia PWM dla pełnej prędkości
#define DUTY_SLOW 1150 // wartość wypełnienia PWM dla mniejszej prędkości - robot porusza się wolniej na zakrętach
#define DUTY_STOP 0 // wartość wypełnienia PWM dla zatrzymania robota - 0 czyli brak ruchu
uint32_t bitSkretu = -1; // zmienna do przechowywania kierunku skrętu, -1 to skręt w prawo, 1 to skręt w lewo
/* USER CODE END PV */
```



Adrian Popielarczyk 21295, Zuzanna Orzechowska 21284

Kod w sekcji USER CODE PFP:

```
/* USER CODE BEGIN PFP */
void dzidaDoPrzodu(void){ // funkcja do jazdy do przodu z pełną prędkością
    //zgodnie z ustawieniami pinów IN1, IN2, IN3, IN4 i działaniem mostka H dla wartości 1010 silniki
    //obracają do przodu
    HAL_GPIO_WritePin(GPIOC, IN1_Pin, GPIO_PIN_SET); // ustawienie IN1 na wysoki stan, IN1 to pin od-
    //powiedzialny za lewy motor
    HAL_GPIO_WritePin(GPIOC, IN2_Pin, GPIO_PIN_RESET); // ustawienie IN2 na niski stan, IN2 to rów-
    //nież pin odpowiedzialny za lewy silnik
    HAL_GPIO_WritePin(GPIOC, IN3_Pin, GPIO_PIN_SET); // ustawienie IN3 na wysoki stan, IN3 to pin od-
    //powiedzialny za prawy motor
    HAL_GPIO_WritePin(GPIOC, IN4_Pin, GPIO_PIN_RESET); // ustawienie IN4 na niski stan, IN4 to rów-
    //nież pin odpowiedzialny za prawy silnik
    __HAL_TIM_SET_COMPARE(&htim2, TIM_CHANNEL_1, duty); // ustawienie wartości wypełnienia PWM dla
    //kanału 1, czyli lewego silnika na pełna moc
    __HAL_TIM_SET_COMPARE(&htim2, TIM_CHANNEL_2, duty); // ustawienie wartości wypełnienia PWM dla
    //kanału 2, czyli prawego silnika na pełna moc
}

void powoli(void){ // funkcja do jazdy do przodu z mniejszą prędkością
    //zgodnie z ustawieniami pinów IN1, IN2, IN3, IN4 i działaniem mostka H dla wartości 1010 silniki
    //obracają do przodu
    HAL_GPIO_WritePin(GPIOC, IN1_Pin, GPIO_PIN_SET); // ustawienie IN1 na wysoki stan, IN1 to pin od-
    //powiedzialny za lewy motor
    HAL_GPIO_WritePin(GPIOC, IN2_Pin, GPIO_PIN_RESET); // ustawienie IN2 na niski stan, IN2 to rów-
    //nież pin odpowiedzialny za lewy silnik
    HAL_GPIO_WritePin(GPIOC, IN3_Pin, GPIO_PIN_SET); // ustawienie IN3 na wysoki stan, IN3 to pin od-
    //powiedzialny za prawy motor
    HAL_GPIO_WritePin(GPIOC, IN4_Pin, GPIO_PIN_RESET); // ustawienie IN4 na niski stan, IN4 to rów-
    //nież pin odpowiedzialny za prawy silnik
    __HAL_TIM_SET_COMPARE(&htim2, TIM_CHANNEL_1, duty); // ustawienie wartości wypełnienia PWM dla
    //kanału 1, czyli lewego silnika na mniejszą moc
    __HAL_TIM_SET_COMPARE(&htim2, TIM_CHANNEL_2, duty); // ustawienie wartości wypełnienia PWM dla
    //kanału 2, czyli prawego silnika na mniejszą moc
}

void skretWLewo(void){ // funkcja do skrętu w lewo
    //zgodnie z ustawieniami pinów IN1, IN2, IN3, IN4 i działaniem mostka H dla wartości 1000 silniki
    //obracają do przodu w lewo
    HAL_GPIO_WritePin(GPIOC, IN1_Pin, GPIO_PIN_SET); // IN1 to lewy motor, ustawienie na wysoki stan
    HAL_GPIO_WritePin(GPIOC, IN2_Pin, GPIO_PIN_RESET); // IN2 to lewy motor, ustawienie na niski stan
    HAL_GPIO_WritePin(GPIOC, IN3_Pin, GPIO_PIN_RESET); // IN3 to prawy motor, ustawienie na niski
    //stan
    HAL_GPIO_WritePin(GPIOC, IN4_Pin, GPIO_PIN_RESET); // IN4 to prawy motor, ustawienie na niski
    //stan
    __HAL_TIM_SET_COMPARE(&htim2, TIM_CHANNEL_1, duty); // ustawienie wartości wypełnienia PWM dla
    //kanału 1, czyli lewego silnika na pełna moc
```



Raport z budowy robota

Adrian Popielarczyk 21295, Zuzanna Orzechowska 21284

```
__HAL_TIM_SET_COMPARE(&htim2, TIM_CHANNEL_2, 0); // ustawienie wartości wypełnienia PWM dla kanału 2, czyli prawego silnika na 0, czyli brak ruchu, dzięki temu motor będzie się obracał w lewą stronę
}

void skretWPravo(void){ // funkcja do skrętu w prawo
    //zgodnie z ustawieniami pinów IN1, IN2, IN3, IN4 i działaniem mostka H dla wartości 0100 silniki obracają do przodu w prawo
    HAL_GPIO_WritePin(GPIOC, IN1_Pin, GPIO_PIN_RESET); // IN1 to lewy motor, ustawienie na niski stan
    HAL_GPIO_WritePin(GPIOC, IN2_Pin, GPIO_PIN_RESET); // IN2 to lewy motor, ustawienie na niski stan
    HAL_GPIO_WritePin(GPIOC, IN3_Pin, GPIO_PIN_SET); // IN3 to prawy motor, ustawienie na wysoki stan
    HAL_GPIO_WritePin(GPIOC, IN4_Pin, GPIO_PIN_RESET); // IN4 to prawy motor, ustawienie na niski stan
    __HAL_TIM_SET_COMPARE(&htim2, TIM_CHANNEL_1, 0); // ustawienie wartości wypełnienia PWM dla kanału 1, czyli lewego silnika na 0, czyli brak ruchu, dzięki temu motor będzie się obracał w prawo
    __HAL_TIM_SET_COMPARE(&htim2, TIM_CHANNEL_2, duty); // ustawienie wartości wypełnienia PWM dla kanału 2, czyli prawego silnika na pełną moc
}

void stop(void){ // funkcja do zatrzymania robota
    //zgodnie z ustawieniami pinów IN1, IN2, IN3, IN4 i działaniem mostka H dla wartości 0000 silniki nie obracają się
    HAL_GPIO_WritePin(GPIOC, IN1_Pin, GPIO_PIN_RESET); //pin IN1 na stan niski
    HAL_GPIO_WritePin(GPIOC, IN2_Pin, GPIO_PIN_RESET); //pin IN2 na stan niski
    HAL_GPIO_WritePin(GPIOC, IN3_Pin, GPIO_PIN_RESET); //pin IN3 na stan niski
    HAL_GPIO_WritePin(GPIOC, IN4_Pin, GPIO_PIN_RESET); //pin IN4 na stan niski
    __HAL_TIM_SET_COMPARE(&htim2, TIM_CHANNEL_1, duty); //wartość wypełnienia PWM dla kanału 1 na pełną moc, motory zatrzymają się "stopniowo"
    __HAL_TIM_SET_COMPARE(&htim2, TIM_CHANNEL_2, duty); //wartość wypełnienia PWM dla kanału 2 na pełną moc, motory zatrzymają się "stopniowo"
}

void lewy90(void){ // funkcja do skrętu w lewo o 90 stopni
    // zgodnie z ustawieniami pinów IN1, IN2, IN3, IN4 i działaniem mostka H dla wartości 0101 silniki obracają do przodu w lewo – o 90 stopni
    HAL_GPIO_WritePin(GPIOC, IN1_Pin, GPIO_PIN_RESET); //pin IN1 na stan niski
    HAL_GPIO_WritePin(GPIOC, IN2_Pin, GPIO_PIN_SET); //pin IN2 na stan wysoki
    HAL_GPIO_WritePin(GPIOC, IN3_Pin, GPIO_PIN_SET); // pin IN3 na stan wysoki
    HAL_GPIO_WritePin(GPIOC, IN4_Pin, GPIO_PIN_RESET); //pin IN4 na stan niski
    __HAL_TIM_SET_COMPARE(&htim2, TIM_CHANNEL_1, duty); //wypełnienie PWM dla lewego silnika na pełną moc
    __HAL_TIM_SET_COMPARE(&htim2, TIM_CHANNEL_2, duty); //wypełnienie PWM dla prawego silnika na pełną moc
    HAL_Delay(300); // opóźnienie 300 ms, aby robot mógł wykonać skręt o 90 stopni
}
```



Raport z budowy robota

Adrian Popielarczyk 21295, Zuzanna Orzechowska 21284

```
void prawy90(void){ // funkcja do skrętu w prawo o 90 stopni
    // zgodnie z ustawieniami pinów IN1, IN2, IN3, IN4 i działaniem mostka H dla wartości 1010 silniki
    // obracają do przodu w prawo - o 90 stopni
    HAL_GPIO_WritePin(GPIOC, IN1_Pin, GPIO_PIN_SET); //pin IN1 na stan wysoki
    HAL_GPIO_WritePin(GPIOC, IN2_Pin, GPIO_PIN_RESET); //pin IN2 na stan niski
    HAL_GPIO_WritePin(GPIOC, IN3_Pin, GPIO_PIN_RESET); // pin IN3 na stan niski
    HAL_GPIO_WritePin(GPIOC, IN4_Pin, GPIO_PIN_SET); //pin IN4 na stan wysoki
    __HAL_TIM_SET_COMPARE(&htim2, TIM_CHANNEL_1, duty); //wypełnienie PWM dla lewego silnika na
    pełną moc
    __HAL_TIM_SET_COMPARE(&htim2, TIM_CHANNEL_2, duty); //wypełnienie PWM dla prawego silnika na
    pełną moc
    HAL_Delay(300); // opóźnienie 300 ms, aby robot mógł wykonać skręt o 90 stopni
}

void searchLine(uint32_t bitSkretu){ // funkcja do poszukiwania linii, jeśli robot nie widzi linii,
    // to będzie szukał linii w kierunku skrętu
    //zgodnie z ustawieniami pinów IN1, IN2, IN3, IN4 i działaniem mostka H dla wartości 1100 silniki
    //obracają do przodu w lewo lub w prawo
    if(bitSkretu == -1){ // jeśli bitSkretu jest równy -1, to robot będzie szukał linii w prawo
        HAL_GPIO_WritePin(GPIOC, IN1_Pin, GPIO_PIN_RESET); //pin IN1 lewego motoru na stan niski
        HAL_GPIO_WritePin(GPIOC, IN2_Pin, GPIO_PIN_SET); //pin IN2 lewego motoru na stan wysoki
        HAL_GPIO_WritePin(GPIOC, IN3_Pin, GPIO_PIN_SET); //pin IN3 prawego motoru na stan wysoki
        HAL_GPIO_WritePin(GPIOC, IN4_Pin, GPIO_PIN_RESET); //pin IN4 prawego motoru na stan niski
        __HAL_TIM_SET_COMPARE(&htim2, TIM_CHANNEL_1, duty); //wartość wypełnienia PWM dla lewego sil-
        //nika na pełną moc
        __HAL_TIM_SET_COMPARE(&htim2, TIM_CHANNEL_2, duty); //wartość wypełnienia PWM dla prawego
        //silnika na pełną moc
    }
    else{ // jeśli bitSkretu jest równy 1, to robot będzie szukał linii w lewo
        HAL_GPIO_WritePin(GPIOC, IN1_Pin, GPIO_PIN_SET); //pin IN1 lewego motoru na stan wysoki
        HAL_GPIO_WritePin(GPIOC, IN2_Pin, GPIO_PIN_RESET); //pin IN2 lewego motoru na stan niski
        HAL_GPIO_WritePin(GPIOC, IN3_Pin, GPIO_PIN_RESET); //pin IN3 prawego motoru na stan niski
        HAL_GPIO_WritePin(GPIOC, IN4_Pin, GPIO_PIN_SET); //pin IN4 prawego motoru na stan wysoki
        __HAL_TIM_SET_COMPARE(&htim2, TIM_CHANNEL_1, duty); // wartość wypełnienia PWM dla lewego sil-
        //nika na pełną moc
        __HAL_TIM_SET_COMPARE(&htim2, TIM_CHANNEL_2, duty); // wartość wypełnienia PWM dla prawego sil-
        //nika na pełną moc
    }
}

/* USER CODE END PFP */
```

Kod w sekcji USER CODE 2 – w tej części zaczęto prace nad działaniem wyświetlacza LCD, działa on „po części”, to znaczy łączy się z płytą oraz ekran się zaświeca, jednak żadne komendy i napisy nie zostają wyświetlone widoczne są jedynie czarne prostokąty w pierwszej linii ekranu.

```
/* USER CODE BEGIN 2 */
    HAL_Delay(2000); // opóźnienie 2 sekund przed uruchomieniem programu odpowiedzialnego za wyście-
    tlacz LCD
```



Adrian Popielarczyk 21295, Zuzanna Orzechowska 21284

```
LCD_Init(); // inicjalizacja wyświetlacza LCD
LCD_SetCursor(0, 0); // ustawienie kurSORA na pierwszą linię i pierwszy znak
LCD_SendString("STM32 + LCD"); // wysłanie napisu do wyświetlacza LCD
LCD_SetCursor(1, 0); // ustawienie kurSORA na drugą linię i pierwszy znak
LCD_SendString("Działa!"); // wysłanie napisu do wyświetlacza LCD

HAL_GPIO_TogglePin(GPIOA, LED_Pin); // dioda Led miga na początku programu, co pokazuje, że ten się uruchomił

while(HAL_GPIO_ReadPin(GPIOC, B1_Pin) == GPIO_PIN_SET) // program czeka na naciśnięcie przycisku aby uruchomić sekwencje jazdy
{
    HAL_Delay(200); // opóźnienie 200 ms, aby uniknąć drgań styków przy naciśnięciu przycisku
}

/* USER CODE END 2 */
```

Kod w sekcji USER CODE WHILE:

```
/* USER CODE BEGIN WHILE */
// Inicjalizacja pinów do sterowania mostkiem H, początkowo wszystkie piny są ustawione na niski stan
HAL_GPIO_WritePin(GPIOC, IN1_Pin, GPIO_PIN_RESET);
HAL_GPIO_WritePin(GPIOC, IN2_Pin, GPIO_PIN_RESET);
HAL_GPIO_WritePin(GPIOC, IN3_Pin, GPIO_PIN_RESET);
HAL_GPIO_WritePin(GPIOC, IN4_Pin, GPIO_PIN_RESET);

HAL_TIM_PWM_Start(&htim2, TIM_CHANNEL_1); // uruchomienie PWM na kanale 1 (lewy silnik)
HAL_TIM_PWM_Start(&htim2, TIM_CHANNEL_2); // uruchomienie PWM na kanale 2 (prawy silnik)

__HAL_TIM_SET_COMPARE(&htim2, TIM_CHANNEL_1, 0); // ustawienie wartości wypełnienia PWM dla kanału 1 (lewy silnik) na 0, czyli brak ruchu
__HAL_TIM_SET_COMPARE(&htim2, TIM_CHANNEL_2, 0); // ustawienie wartości wypełnienia PWM dla kanału 2 (prawy silnik) na 0, czyli brak ruchu
while (1) // główna pętla programu, robot porusza się do momentu az nie zostanie ręcznie wyłączony
{
    lewy = HAL_GPIO_ReadPin(IR1_GPIO_Port, IR1_Pin); // odczyt stanu lewego czujnika - czy linia pod nim jest wykrywalna
    srodek = HAL_GPIO_ReadPin(IR2_GPIO_Port, IR2_Pin); // odczyt stanu środkowego czujnika - czy linia pod nim jest wykrywalna
    prawy = HAL_GPIO_ReadPin(IR3_GPIO_Port, IR3_Pin); // odczyt stanu prawego czujnika - czy linia pod nim jest wykrywalna

    if (lewy && srodek && prawy) { // jeśli wszystkie czujniki widzą linię, to robot jedzie do przodu z pełną prędkością, pozwoli to na staranne zatrzymanie
        duty = DUTY_STOP; // prędkość na 0, aby robot mógł się zatrzymać
        powoli(); //stop
    }
```



Adrian Popielarczyk 21295, Zuzanna Orzechowska 21284

```
else if (lewy && srodek && !prawy) { // jeśli lewy i środkowy czujnik widzą linię, a prawy nie, to robot skręca w lewo
    duty = DUTY_SLOW; // ustawienie mniejszej prędkości, aby robot mógł skrącić
    skretWLewo(); // wywołanie funkcji do skrętu w lewo
    bitSkretu=1; // ustawienie bitu skrętu na 1, co oznacza skręt w lewo
}
else if (prawy && srodek && !lewy) { // jeśli prawy i środkowy czujnik widzą linię, a lewy nie, to robot skręca w prawo
    duty = DUTY_SLOW; // ustawienie mniejszej prędkości, aby robot mógł skrącić
    skretWPravo(); // wywołanie funkcji do skrętu w prawo
    bitSkretu=-1; // ustawienie bitu skrętu na -1, co oznacza skręt w prawo
}
else if (srodek && !lewy && !prawy) { // jeśli tylko środkowy czujnik widzi linię, to robot jedzie do przodu z pełną prędkością
    duty = DUTY_FAST; // ustawienie pełnej prędkości, aby robot mógł jechać do przodu
    dzidaDoPrzodu(); // wywołanie funkcji do jazdy do przodu z pełną prędkością
}
else if (lewy && !srodek && !prawy) { // jeśli tylko lewy czujnik widzi linię, to robot skręca w lewo
    duty = DUTY_SLOW; // ustawienie mniejszej prędkości, aby robot mógł skrącić
    skretWLewo(); // wywołanie funkcji do skrętu w lewo
    bitSkretu=1; // ustawienie bitu skrętu na 1, co oznacza skręt w lewo
}
else if (prawy && !srodek && !lewy) { // jeśli tylko prawy czujnik widzi linię, to robot skręca w prawo
    duty = DUTY_SLOW; // ustawienie mniejszej prędkości, aby robot mógł skrącić
    skretWPravo(); // wywołanie funkcji do skrętu w prawo
    bitSkretu=-1; // ustawienie bitu skrętu na -1, co oznacza skręt w prawo
}
else { // jeśli żaden czujnik nie widzi linii, to robot będzie szukał linii w kierunku skrętu
    duty = DUTY_SLOW; // ustawienie mniejszej prędkości, aby robot mógł szukać linii
    searchLine(bitSkretu); // wywołanie funkcji do szukania linii w kierunku skrętu
}

HAL_Delay(100); // opóźnienie 100 ms, aby robot mógł reagować na zmiany stanu czujników
/* USER CODE END WHILE */

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

Poniżej przedstawiono kod programu dla pliku `Lcd_hd44780.c` zawierający kod obsługujący wyświetlacz:

```
// lcd_hd44780.c
#include "lcd_hd44780.h" //implementacja własnej biblioteki obsługującej wyświetlacz LCD HD44780
#include "stm32l0xx_hal.h" //biblioteka HAL dla płytki STM32L0
#include "string.h" //biblioteka do obsługi łańcuchów znaków
```



Adrian Popielarczyk 21295, Zuzanna Orzechowska 21284

```
static void LCD_EnablePulse(void) { // Funkcja do generowania impulsu na linii Enable
    HAL_GPIO_WritePin(LCD_EN_GPIO_Port, LCD_EN_Pin, GPIO_PIN_SET); // Ustawienie linii Enable na stan wysoki, aby zainicjować zapis danych
    HAL_Delay(1); // zwiększyony czas
    HAL_GPIO_WritePin(LCD_EN_GPIO_Port, LCD_EN_Pin, GPIO_PIN_RESET); // Ustawienie linii Enable na stan niski, aby zakończyć zapis danych
    HAL_Delay(1); // zwiększyony czas
}

static void LCD_Write4Bits(uint8_t data) { // Funkcja do zapisu 4 bitów danych na wyświetlaczu LCD
    HAL_GPIO_WritePin(LCD_D4_GPIO_Port, LCD_D4_Pin, ((data >> 0) & 0x01) ? GPIO_PIN_SET : GPIO_PIN_RESET); // Ustawienie linii D4 na stan wysoki lub niski w zależności od wartości bitu
    HAL_GPIO_WritePin(LCD_D5_GPIO_Port, LCD_D5_Pin, ((data >> 1) & 0x01) ? GPIO_PIN_SET : GPIO_PIN_RESET); // Ustawienie linii D5 na stan wysoki lub niski w zależności od wartości bitu
    HAL_GPIO_WritePin(LCD_D6_GPIO_Port, LCD_D6_Pin, ((data >> 2) & 0x01) ? GPIO_PIN_SET : GPIO_PIN_RESET); // Ustawienie linii D6 na stan wysoki lub niski w zależności od wartości bitu
    HAL_GPIO_WritePin(LCD_D7_GPIO_Port, LCD_D7_Pin, ((data >> 3) & 0x01) ? GPIO_PIN_SET : GPIO_PIN_RESET); // Ustawienie linii D7 na stan wysoki lub niski w zależności od wartości bitu
    LCD_EnablePulse(); // Wywołanie funkcji do wygenerowania impulsu na linii Enable, co powoduje zapis danych do wyświetlacza LCD
}

static void LCD_Send(uint8_t data, uint8_t rs) { // Funkcja do wysyłania danych lub poleceń do wyświetlacza LCD
    HAL_GPIO_WritePin(LCD_RS_GPIO_Port, LCD_RS_Pin, rs); // Ustawienie linii RS na stan wysoki (dane) lub niski (polecenie)
    LCD_Write4Bits(data >> 4); // Wysłanie wyższych 4 bitów danych lub polecenia
    LCD_Write4Bits(data & 0x0F); // Wysłanie niższych 4 bitów danych lub polecenia
}

void LCD_SendCommand(uint8_t cmd) { // Funkcja do wysyłania polecenia do wyświetlacza LCD
    LCD_Send(cmd, 0); // Wywołanie funkcji do wysłania polecenia, ustawiając linię RS na stan niski (polecenie)
}

void LCD_SendData(uint8_t data) { // Funkcja do wysyłania danych (znaków) do wyświetlacza LCD
    LCD_Send(data, 1); // Wywołanie funkcji do wysłania danych, ustawiając linię RS na stan wysoki (dane)
}

void LCD_Clear(void) { // Funkcja do czyszczenia wyświetlacza LCD
    LCD_SendCommand(0x01); // Wysłanie polecenia do czyszczenia wyświetlacza
    HAL_Delay(2); // Opóźnienie, aby dać czas na wykonanie polecenia czyszczenia
}

void LCD_SetCursor(uint8_t row, uint8_t col) { // Funkcja do ustawiania kurSORA na określonej pozycji na wyświetlaczu LCD
    uint8_t addr = (row == 0) ? 0x00 : 0x40; // Adres początkowy wiersza (0x00 dla pierwszego wiersza, 0x40 dla drugiego)
```



Adrian Popielarczyk 21295, Zuzanna Orzechowska 21284

```
LCD_SendCommand(0x80 | (addr + col)); // Wysłanie polecenia do ustawienia kurSORA, dodając adres wiersza i kolumny
}

void LCD_SendString(char* str) { // Funkcja do wysyłania łańcucha znaków do wyświetlacza LCD
    while(*str) { // Pętla, która iteruje przez każdy znak w łańcuchu
        LCD_SendData(*str++);
    }
}

void LCD_Init(void) { // Funkcja do inicjalizacji wyświetlacza LCD
    HAL_Delay(100); // Opóźnienie na rozpoczęcie działania wyświetlacza LCD
    HAL_GPIO_WritePin(LCD_RS_GPIO_Port, LCD_RS_Pin, GPIO_PIN_RESET); // Ustawienie linii RS na stan niski, aby wysyłać polecenia
    HAL_GPIO_WritePin(LCD_EN_GPIO_Port, LCD_EN_Pin, GPIO_PIN_RESET); // Ustawienie linii Enable na stan niski, aby zakończyć poprzednie operacje

    // Inicjalizacja w trybie 8-bitowym
    LCD_Write4Bits(0x03);
    HAL_Delay(5);
    LCD_Write4Bits(0x03);
    HAL_Delay(5);
    LCD_Write4Bits(0x03);
    HAL_Delay(5);
    LCD_Write4Bits(0x02); // przejście do trybu 4-bitowego
    HAL_Delay(5);

    LCD_SendCommand(0x28); // Funkcja do ustawienia trybu pracy wyświetlacza (4-bitowy, 2 linie, 5x8 znaków)
    HAL_Delay(2);
    LCD_SendCommand(0x0C); // Funkcja do włączenia wyświetlacza i wyłączenia kurSORA
    HAL_Delay(2);
    LCD_SendCommand(0x06); // Funkcja do ustawienia kierunku przesuwania kurSORA (w prawo)
    HAL_Delay(2);
    LCD_Clear(); // Funkcja do czyszczenia wyświetlacza LCD
}
```

Poniżej znajduje się opis kodu w własnej bibliotece do wyświetlacza – plik lcd_hd44780.h, zawiera on głównie deklaracje odpowiednio podłączonych pinów oraz inicjalizację wykorzystanych funkcji.:

```
#ifndef __LCD_HD44780_H__ // dyrektywa preprocesora, aby uniknąć wielokrotnego dołączenia pliku nagłówkowego
#define __LCD_HD44780_H__ // definicja makra __LCD_HD44780_H__, które jest używane do ochrony przed wielokrotnym dołączeniem tego pliku nagłówkowego

#include "stm32l0xx_hal.h" // dołączenie pliku nagłówkowego HAL dla STM32L0, który zawiera definicje i funkcje potrzebne do pracy z mikrokontrolerem
```

Adrian Popielarczyk 21295, Zuzanna Orzechowska 21284

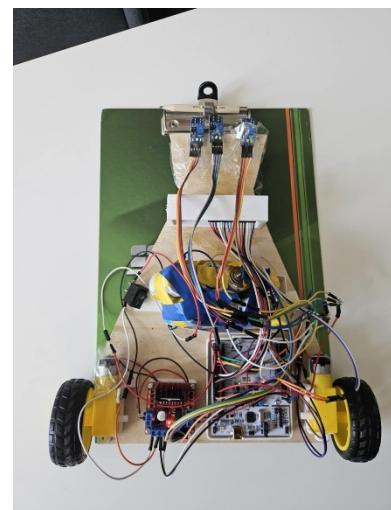
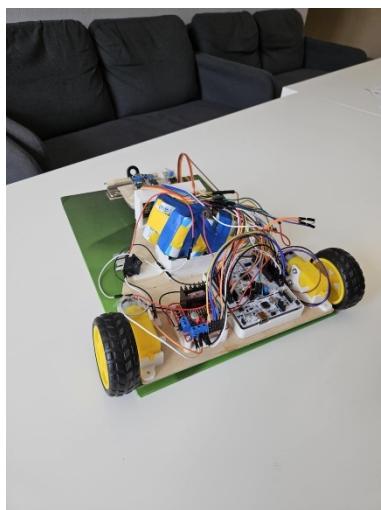
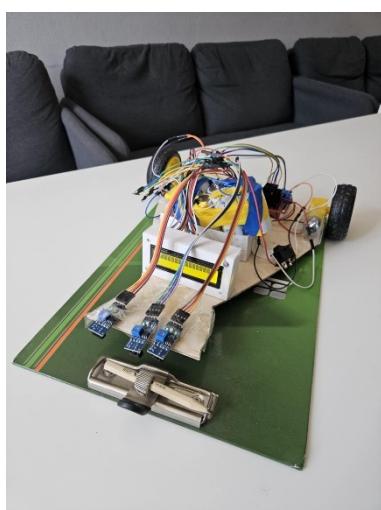
```
// definicje pinów do podłączenia wyświetlacza LCD HD44780
#define LCD_RS_GPIO_Port    GPIOB // port GPIO dla linii RS (Register Select)
#define LCD_RS_Pin           GPIO_PIN_11 // pin GPIO dla linii RS (Register Select)
#define LCD_EN_GPIO_Port     GPIOB // port GPIO dla linii EN (Enable)
#define LCD_EN_Pin           GPIO_PIN_10 // pin GPIO dla linii EN (Enable)

// definicje pinów danych D4, D5, D6, D7
#define LCD_D4_GPIO_Port     GPIOA // port GPIO dla linii D4
#define LCD_D4_Pin            GPIO_PIN_10 // pin GPIO dla linii D4
#define LCD_D5_GPIO_Port     GPIOA // port GPIO dla linii D5
#define LCD_D5_Pin            GPIO_PIN_11 // pin GPIO dla linii D5
#define LCD_D6_GPIO_Port     GPIOA // port GPIO dla linii D6
#define LCD_D6_Pin            GPIO_PIN_12 // pin GPIO dla linii D6
#define LCD_D7_GPIO_Port     GPIOA // port GPIO dla linii D7
#define LCD_D7_Pin            GPIO_PIN_13 // pin GPIO dla linii D7

// funkcje do obsługi wyświetlacza LCD HD44780
void LCD_Init(void); // inicjalizacja wyświetlacza LCD
void LCD_SendCommand(uint8_t); // wysyłanie polecenia do wyświetlacza LCD
void LCD_SendData(uint8_t); // wysyłanie danych (znaków) do wyświetlacza LCD
void LCD_SendString(char*); // wysyłanie łańcucha znaków do wyświetlacza LCD
void LCD_SetCursor(uint8_t row, uint8_t col); // ustawianie kurSORA na określonej pozycji na wyświetlaczu LCD
void LCD_Clear(void); // czyszczenie wyświetlacza LCD

#endif
```

Zdjęcia opracowanego robota:





Adrian Popielarczyk 21295, Zuzanna Orzechowska 21284

Wnioski:

Robot poprawnie pokonuje trasę – również te wytyczone pod kątem 90 stopni.

Postępy dla Milestone III na kolejnej stronie →

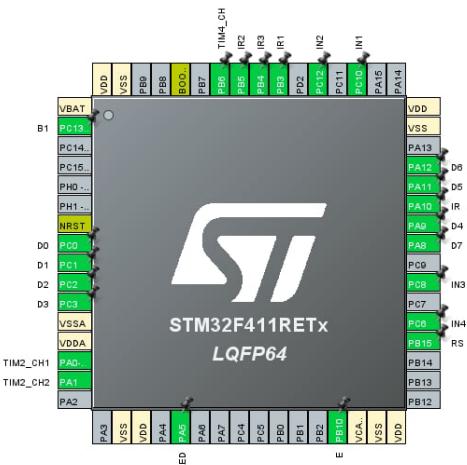


Adrian Popielarczyk 21295, Zuzanna Orzechowska 21284

Milestone III – 03.06.2025

Oprogramowanie sterujące

Konfiguracja pinów:



Poniżej znajduje się opis fragmentów głównego programu – main.c. Kod opatrzono opisami w postaci komentarzy:

Kod w sekcji USER CODE INCLUDES:

```
/* USER CODE BEGIN Includes */  
#include <string.h>  
#include "stm32f4xx_hal.h"  
#include <lcd_hd44780.h> // biblioteka LCD HD44780  
#include <stdio.h>  
#include <stdbool.h> //biblioteka do booleanów  
/* USER CODE END Includes */
```

Kod w sekcji USER CODE PD:

```
/* USER CODE BEGIN PD */  
#define IR_RX_BUFFER_SIZE 100 // Maksymalny rozmiar bufora do przechwytywania impulsów IR  
  
#ifndef IR_Pin // Definicja pinu IR  
#define IR_Pin GPIO_PIN_10 //pin IR jest podłączony do PA10  
#endif //  
#ifndef IR_GPIO_Port // Definicja portu IR  
#define IR_GPIO_Port GPIOA  
#endif  
  
#define NEC_TIMEOUT_US 100000 //maksymalny czas oczekiwania na ramkę IR w mikrosekundach czyli 100ms
```



Adrian Popielarczyk 21295, Zuzanna Orzechowska 21284

```
#define NEC_FRAME_START_MIN_GAP_US 12000 // minimalny czas przerwy między ramkami IR w mikrosekundach czyli 12ms

#define NEC_AGC_MARK_MIN_US 8000 // minimalny czas impulsu AGC w mikrosekundach czyli 8ms
#define NEC_AGC_MARK_MAX_US 10000 // maksymalny czas impulsu AGC w mikrosekundach czyli 10ms
#define NEC_AGC_SPACE_MIN_US 4000 // minimalny czas przerwy AGC w mikrosekundach czyli 4ms
#define NEC_AGC_SPACE_MAX_US 5000 // maksymalny czas przerwy AGC w mikrosekundach czyli 5ms

#define NEC_BIT_MARK_MIN_US 400 // minimalny czas impulsu bitu w mikrosekundach czyli 0.4ms
#define NEC_BIT_MARK_MAX_US 750 // maksymalny czas impulsu bitu w mikrosekundach czyli 0.75ms

#define NEC_BIT_0_SPACE_MIN_US 400 // minimalny czas przerwy bitu 0 w mikrosekundach czyli 0.4ms
#define NEC_BIT_0_SPACE_MAX_US 750 // maksymalny czas przerwy bitu 0 w mikrosekundach czyli 0.75ms

#define NEC_BIT_1_SPACE_MIN_US 1500 // minimalny czas przerwy bitu 1 w mikrosekundach czyli 1.5ms
#define NEC_BIT_1_SPACE_MAX_US 1900 // maksymalny czas przerwy bitu 1 w mikrosekundach czyli 1.9ms

#define NEC_PULSE_COUNT 67 // liczba impulsów w ramce IR NEC (32 bity danych + 1 bit parzystości + 3 bity startowe + 1 bit stopu)

#define NEC_REPEAT_SPACE_MIN_US 2000 // minimalny czas przerwy powtarzającej ramki IR w mikrosekundach czyli 2ms
#define NEC_REPEAT_SPACE_MAX_US 2500 // maksymalny czas przerwy powtarzającej ramki IR w mikrosekundach czyli 2.5ms
#define NEC_REPEAT_PULSE_COUNT 3 // liczba impulsów w powtarzającej ramce IR NEC (2 bity startowe + 1 bit stopu)

#define TIM_IR_PRESCALER (15) // Preskaler dla timera IR, aby uzyskać 1us rozdzielcość, gdzie us to mikrosekundy

#define DISCRETE_CMD_MIN_INTERVAL_MS 400 // minimalny czas między powtarzającymi się poleceniami w trybie zdalnym w milisekundach
/* USER CODE END PD */
```

Kod w sekcji USER CODE PV:

```
/* USER CODE BEGIN PV */
uint32_t duty = 1400;
#define DUTY_FAST 1400
#define DUTY_SLOW 1150
#define DUTY_STOP 0
int bitSkretu = 1; // 1 - skret w lewo, -1 - skret w prawo, 0 - jazda na wprost

volatile uint8_t control_mode = 0; // 0 - tryb automatyczny, 1 - tryb zdalny

volatile uint32_t last_edge_time = 0; //czas ostatniego zbocza w mikrosekundach
volatile uint32_t pulse_widths[IR_RX_BUFFER_SIZE]; //bufor do przechwytywania impulsów IR
volatile uint8_t pulse_index = 0; //indeks aktualnego impulsu w buforze
volatile bool ir_frame_ready = false; //flaga informująca, czy ramka IR jest gotowa do przetwarzania
```



Adrian Popielarczyk 21295, Zuzanna Orzechowska 21284

```
volatile bool capture_started = false; // flaga informująca, czy rozpoczęto przechwytywanie impulsów IR

volatile bool ir_repeat_detected = false; // flaga informująca, czy wykryto powtarzającą ramkę IR
volatile uint32_t last_good_ir_code = 0xFFFFFFFF; // ostatni poprawny kod IR, który został przetworzony

volatile bool is_moving_remotely = false; // flaga informująca, czy robot jest w ruchu w trybie zdalnym
volatile uint32_t last_remote_movement_command_time = 0; // czas ostatniego polecenia ruchu w trybie zdalnym
#define REMOTE_MOVEMENT_TIMEOUT 300 // czas w milisekundach, po którym robot zatrzyma się, jeśli nie otrzyma kolejnego polecenia ruchu

uint32_t last_processed_discrete_ir_code = 0xFFFFFFFF; // ostatni przetworzony kod IR dla poleceń dyskretnych
uint32_t last_processed_discrete_ir_time = 0; // czas ostatniego przetworzenia polecenia dyskretnego, polecenie dyskretne nie jest powtarzane, np. zmiana trybu pracy

static bool lcd_initialized = false; // flaga informująca, czy LCD zostało zainicjalizowane

typedef struct //struktura do przechowywania nuty
{
    uint16_t frequency; // Hz to częstotliwość nuty
    uint16_t duration; // ms to czas trwania nuty w milisekundach
} Note; // definicja struktury Note

//definicje częstotliwości nut w Hz
#define NOTE_C4 262 // C4 to 262 Hz
#define NOTE_D4 294 // D4 to 294 Hz
#define NOTE_E4 330 // E4 to 330 Hz
#define NOTE_F4 349 // F4 to 349 Hz
#define NOTE_G4 392 // G4 to 392 Hz
#define NOTE_A4 440 // A4 to 440 Hz
#define NOTE_B4 494 // B4 to 494 Hz
#define NOTE_C5 523 // C5 to 523 Hz
#define NOTE_D5 587 // D5 to 587 Hz
#define NOTE_E5 659 // E5 to 659 Hz
#define NOTE_F5 698 // F5 to 698 Hz
#define NOTE_G5 784 // G5 to 784 Hz
#define NOTE_A5 880 // A5 to 880 Hz
#define NOTE_B5 988 // B5 to 988 Hz
#define NOTE_C6 1047 // C6 to 1047 Hz
#define NOTE_D6 1175 // D6 to 1175 Hz
#define NOTE_REST 0 // NOTE_REST to 0 Hz, czyli cisza
//wraz ze wzrostem częstotliwości nut, rośnie ich wysokość

Note happy_bounce[] = { // tablica nut do melodii "Happy Bounce"
```



Adrian Popielarczyk 21295, Zuzanna Orzechowska 21284

```
{NOTE_C5, 300}, {NOTE_REST, 50}, {NOTE_E5, 300}, {NOTE_REST, 50}, {NOTE_G5, 300}, {NOTE_REST, 50}, {NOTE_C6, 600}, {NOTE_REST, 100},  
// pierwsza część melodii  
{NOTE_B5, 300}, // czas trwania nuty B5 to 300 ms, analogicznie do reszty nut w kodzie  
{NOTE_REST, 50}, // czas trwania ciszy to 50 ms, analogicznie do reszty nut w kodzie  
{NOTE_G5, 300},  
{NOTE_REST, 50},  
{NOTE_E5, 300},  
{NOTE_REST, 50},  
{NOTE_C5, 600},  
{NOTE_REST, 100},  
  
{NOTE_D5, 300},  
{NOTE_REST, 50},  
{NOTE_F5, 300},  
{NOTE_REST, 50},  
{NOTE_A5, 300},  
{NOTE_REST, 50},  
{NOTE_B5, 300},  
{NOTE_REST, 50},  
{NOTE_C6, 600},  
{NOTE_REST, 100},  
  
{NOTE_C6, 150},  
{NOTE_B5, 150},  
{NOTE_A5, 150},  
{NOTE_G5, 150},  
{NOTE_F5, 150},  
{NOTE_E5, 150},  
{NOTE_D5, 150},  
{NOTE_C5, 600},  
{NOTE_REST, 200},  
  
{NOTE_REST, 0}};  
  
/* USER CODE END PV */
```

Kod w sekcji USER CODE PFP:

```
/* USER CODE BEGIN PFP */  
void play_note(uint16_t freq, uint16_t duration_ms) // funkcja do odtwarzania nuty, freq to częstotliwość nuty w Hz, duration_ms to czas trwania nuty w milisekundach  
{  
    if (freq == 0) // jeśli częstotliwość jest równa 0, to odtwarzamy ciszę  
    {  
        HAL_TIM_PWM_Stop(&htim4, TIM_CHANNEL_1); // zatrzymujemy PWM na kanale 1 timera 4  
        HAL_Delay(duration_ms); // czekamy przez czas trwania nuty  
        return;  
    }
```



Adrian Popielarczyk 21295, Zuzanna Orzechowska 21284

```
uint32_t tim_clock = 16000000; // zegar timera 4 to 16 MHz
uint16_t prescaler = 71; // preskaler timera 4, aby uzyskać 1 MHz zegar timera
uint32_t period = (tim_clock / (prescaler + 1)) / freq - 1; // okres timera 4, aby uzyskać częstotliwość nuty, wzór oznacza // okres timera jako liczba cykli zegara podzielona przez częstotliwość nuty minus 1

__HAL_TIM_SET_PRESCALER(&htim4, prescaler); // ustawiamy preskaler timera 4
__HAL_TIM_SET_AUTORELOAD(&htim4, period); // ustawiamy okres timera 4
__HAL_TIM_SET_COMPARE(&htim4, TIM_CHANNEL_1, period / 2); // ustawiamy wartość porównania timera 4, aby uzyskać 50% wypełnienia PWM
HAL_TIM_PWM_Start(&htim4, TIM_CHANNEL_1); // uruchamiamy PWM na kanale 1 timera 4

HAL_Delay(duration_ms); // czekamy przez czas trwania nuty

HAL_TIM_PWM_Stop(&htim4, TIM_CHANNEL_1); // zatrzymujemy PWM na kanale 1 timera 4
}

void play_melody(Note *melody) // funkcja do odtwarzania melodii, melody to wskaźnik do tablicy nut
{
    for (int i = 0; melody[i].duration > 0; i++) // iterujemy przez tablicę nut, aż do napotkania nuty z czasem trwania 0, czyli końca melodii
    {
        play_note(melody[i].frequency, melody[i].duration); // odtwarzamy nutę z częstotliwością i czasem trwania
    }
}

uint32_t decode_nec(volatile uint32_t *pulses, uint8_t count); // funkcja do dekodowania ramki IR NEC, przyjmuje wskaźnik do bufora impulsów i liczbę impulsów

// Funkcje do sterowania silnikami, zostały opisane w kodzie do poprzedniego Milestone II
void dzidaDoPrzodu(void)
{
    HAL_GPIO_WritePin(GPIOC, IN1_Pin, GPIO_PIN_SET);
    HAL_GPIO_WritePin(GPIOC, IN2_Pin, GPIO_PIN_RESET);
    HAL_GPIO_WritePin(GPIOC, IN3_Pin, GPIO_PIN_SET);
    HAL_GPIO_WritePin(GPIOC, IN4_Pin, GPIO_PIN_RESET);
    __HAL_TIM_SET_COMPARE(&htim2, TIM_CHANNEL_1, duty);
    __HAL_TIM_SET_COMPARE(&htim2, TIM_CHANNEL_2, duty);
}

void doTylu(void)
{
    HAL_GPIO_WritePin(GPIOC, IN1_Pin, GPIO_PIN_RESET);
    HAL_GPIO_WritePin(GPIOC, IN2_Pin, GPIO_PIN_SET);
    HAL_GPIO_WritePin(GPIOC, IN3_Pin, GPIO_PIN_RESET);
    HAL_GPIO_WritePin(GPIOC, IN4_Pin, GPIO_PIN_SET);
    __HAL_TIM_SET_COMPARE(&htim2, TIM_CHANNEL_1, duty);
    __HAL_TIM_SET_COMPARE(&htim2, TIM_CHANNEL_2, duty);
}
```



Adrian Popielarczyk 21295, Zuzanna Orzechowska 21284

```
}

void skretWLewo(void)
{
    HAL_GPIO_WritePin(GPIOC, IN1_Pin, GPIO_PIN_RESET);
    HAL_GPIO_WritePin(GPIOC, IN2_Pin, GPIO_PIN_RESET);
    HAL_GPIO_WritePin(GPIOC, IN3_Pin, GPIO_PIN_SET);
    HAL_GPIO_WritePin(GPIOC, IN4_Pin, GPIO_PIN_RESET);
    __HAL_TIM_SET_COMPARE(&htim2, TIM_CHANNEL_1, duty);
    __HAL_TIM_SET_COMPARE(&htim2, TIM_CHANNEL_2, duty);
}

void skretWPrawo(void)
{
    HAL_GPIO_WritePin(GPIOC, IN1_Pin, GPIO_PIN_SET);
    HAL_GPIO_WritePin(GPIOC, IN2_Pin, GPIO_PIN_RESET);
    HAL_GPIO_WritePin(GPIOC, IN3_Pin, GPIO_PIN_RESET);
    HAL_GPIO_WritePin(GPIOC, IN4_Pin, GPIO_PIN_RESET);
    __HAL_TIM_SET_COMPARE(&htim2, TIM_CHANNEL_1, duty);
    __HAL_TIM_SET_COMPARE(&htim2, TIM_CHANNEL_2, duty);
}

void stop(void)
{
    HAL_GPIO_WritePin(GPIOC, IN1_Pin, GPIO_PIN_RESET);
    HAL_GPIO_WritePin(GPIOC, IN2_Pin, GPIO_PIN_RESET);
    HAL_GPIO_WritePin(GPIOC, IN3_Pin, GPIO_PIN_RESET);
    HAL_GPIO_WritePin(GPIOC, IN4_Pin, GPIO_PIN_RESET);
    __HAL_TIM_SET_COMPARE(&htim2, TIM_CHANNEL_1, 0);
    __HAL_TIM_SET_COMPARE(&htim2, TIM_CHANNEL_2, 0);
}

void lewy90(void)
{
    HAL_GPIO_WritePin(GPIOC, IN1_Pin, GPIO_PIN_RESET);
    HAL_GPIO_WritePin(GPIOC, IN2_Pin, GPIO_PIN_SET);
    HAL_GPIO_WritePin(GPIOC, IN3_Pin, GPIO_PIN_SET);
    HAL_GPIO_WritePin(GPIOC, IN4_Pin, GPIO_PIN_RESET);
    __HAL_TIM_SET_COMPARE(&htim2, TIM_CHANNEL_1, duty);
    __HAL_TIM_SET_COMPARE(&htim2, TIM_CHANNEL_2, duty);
}

void prawy90(void)
{
    HAL_GPIO_WritePin(GPIOC, IN1_Pin, GPIO_PIN_SET);
    HAL_GPIO_WritePin(GPIOC, IN2_Pin, GPIO_PIN_RESET);
    HAL_GPIO_WritePin(GPIOC, IN3_Pin, GPIO_PIN_RESET);
    HAL_GPIO_WritePin(GPIOC, IN4_Pin, GPIO_PIN_SET);
    __HAL_TIM_SET_COMPARE(&htim2, TIM_CHANNEL_1, duty);
```



Adrian Popielarczyk 21295, Zuzanna Orzechowska 21284

```
__HAL_TIM_SET_COMPARE(&htim2, TIM_CHANNEL_2, duty);  
}  
  
// Funkcja do szukania linii, jeśli robot nie wykrywa linii, to będzie ją szukał  
void searchLine(int lastTurnDirection) // funkcja do szukania linii, lastTurnDirection to kierunek  
ostatniego skrętu, 1 - lewo, -1 - prawo, 0 - jazda na wprost  
{  
    uint32_t search_duty = DUTY_SLOW; // ustawiamy prędkość szukania linii na wolno  
  
    if (lastTurnDirection >= 0) // jeśli ostatni skręt był w lewo lub jazda na wprost  
    {  
        HAL_GPIO_WritePin(GPIOC, IN1_Pin, GPIO_PIN_SET); // ustawiamy IN1 na wysoki, motor lewy bę-  
dzie jechał do przodu  
        HAL_GPIO_WritePin(GPIOC, IN2_Pin, GPIO_PIN_RESET); // ustawiamy IN2 na niski, motor lewy  
będzie jechał do przodu  
        HAL_GPIO_WritePin(GPIOC, IN3_Pin, GPIO_PIN_RESET); // ustawiamy IN3 na niski, motor prawy  
będzie jechał do tyłu  
        HAL_GPIO_WritePin(GPIOC, IN4_Pin, GPIO_PIN_SET); // ustawiamy IN4 na wysoki, motor prawy  
będzie jechał do tyłu  
    }  
    else // jeśli ostatni skręt był w prawo  
    {  
        HAL_GPIO_WritePin(GPIOC, IN1_Pin, GPIO_PIN_RESET); // ustawiamy IN1 na niski, motor lewy  
będzie jechał do tyłu  
        HAL_GPIO_WritePin(GPIOC, IN2_Pin, GPIO_PIN_SET); // ustawiamy IN2 na wysoki, motor lewy bę-  
dzie jechał do tyłu  
        HAL_GPIO_WritePin(GPIOC, IN3_Pin, GPIO_PIN_SET); // ustawiamy IN3 na wysoki, motor prawy  
będzie jechał do przodu  
        HAL_GPIO_WritePin(GPIOC, IN4_Pin, GPIO_PIN_RESET); // ustawiamy IN4 na niski, motor prawy  
będzie jechał do przodu  
    }  
    __HAL_TIM_SET_COMPARE(&htim2, TIM_CHANNEL_1, search_duty); // ustawiamy wypełnienie PWM dla le-  
wego motoru  
    __HAL_TIM_SET_COMPARE(&htim2, TIM_CHANNEL_2, search_duty); // ustawiamy wypełnienie PWM dla  
prawego motoru  
}  
/* USER CODE END PFP */
```

Kod w sekcji USER CODE BEGIN 0:

```
/* USER CODE BEGIN 0 */  
  
void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin) // funkcja wywoływana po przerwaniu zewnętrznym,  
czyli po wykryciu zbocza na pinie IR  
{  
    if (GPIO_Pin == IR_Pin) // sprawdzamy, czy przerwanie pochodzi z pinu IR  
    {  
        if (ir_frame_ready || ir_repeat_detected) // jeśli ramka IR jest już gotowa lub wykryto po-  
wtarzającą ramkę, to ignorujemy to zbocze  
        return;  
    }  
}
```



Adrian Popielarczyk 21295, Zuzanna Orzechowska 21284

```
uint32_t now = __HAL_TIM_GET_COUNTER(&htim3); // pobieramy aktualny czas z timera 3, który jest skonfigurowany do liczenia czasu w mikrosekundach
uint32_t duration = (uint16_t)(now - last_edge_time); // obliczamy czas trwania impulsu jako różnicę między aktualnym czasem a czasem ostatniego zbocza

if (duration > NEC_FRAME_START_MIN_GAP_US) // jeśli czas trwania impulsu jest większy niż minimalny czas przerwy między ramkami IR
{
    pulse_index = 0; // resetujemy indeks bufora impulsów
    capture_started = true; // rozpoczynamy przechwytywanie impulsów IR
    last_edge_time = now; // aktualizujemy czas ostatniego zbocza
    return; // kończymy funkcję, ponieważ rozpoczęto przechwytywanie impulsów IR
}

if (capture_started) // jeśli przechwytywanie impulsów IR zostało rozpoczęte
{
    if (pulse_index < IR_RX_BUFFER_SIZE) // sprawdzamy, czy indeks bufora impulsów nie przekracza rozmiaru bufora
    {
        pulse_widths[pulse_index++] = duration; // zapisujemy czas trwania impulsu do bufora i zwiększamy indeks
    }
    else // jeśli indeks bufora przekracza rozmiar bufora
    {
        capture_started = false; // zatrzymujemy przechwytywanie impulsów IR
        pulse_index = 0; // resetujemy indeks bufora impulsów
    }

    if (capture_started) // jeśli przechwytywanie impulsów IR nadal trwa
    {
        if (pulse_index == NEC_REPEAT_PULSE_COUNT) // jeśli przechwycono wystarczającą liczbę impulsów dla powtarzającej ramki IR
        {
            if ((pulse_widths[0] >= NEC_AGC_MARK_MIN_US && pulse_widths[0] <= NEC_AGC_MARK_MAX_US) && // sprawdzamy, czy pierwszy impuls jest w zakresie AGC
                (pulse_widths[1] >= NEC_REPEAT_SPACE_MIN_US && pulse_widths[1] <= NEC_REPEAT_SPACE_MAX_US) && // sprawdzamy, czy drugi impuls jest w zakresie powtarzającej ramki
                (pulse_widths[2] >= NEC_BIT_MARK_MIN_US && pulse_widths[2] <= NEC_BIT_MARK_MAX_US)) // sprawdzamy, czy trzeci impuls jest w zakresie bitu
            {
                ir_repeat_detected = true; // ustawiamy flagę wykrycia powtarzającej ramki IR
                capture_started = false; // zatrzymujemy przechwytywanie impulsów IR
            }
        }

        if (!ir_repeat_detected && pulse_index >= NEC_PULSE_COUNT) // jeśli nie wykryto powtarzającej ramki IR i przechwycono wystarczającą liczbę impulsów dla pełnej ramki IR
    }
}
```



Adrian Popielarczyk 21295, Zuzanna Orzechowska 21284

```
{  
    ir_frame_ready = true; // ustawiamy flagęgotowości ramki IR do przetworzenia  
    capture_started = false; // zatrzymujemy przechwytywanie impulsów IR  
}  
}  
}  
}  
last_edge_time = now; // aktualizujemy czas ostatniego zbocza  
}  
}  
  
uint32_t decode_nec(volatile uint32_t *pulses, uint8_t count) // funkcja do dekodowania ramki IR  
NEC, przyjmuje wskaźnik do bufora impulsów i liczbę impulsów  
{  
    if (count < NEC_PULSE_COUNT) // sprawdzamy, czy liczba impulsów jest mniejsza niż minimalna  
    liczba impulsów dla ramki IR NEC  
        return 0xFFFFFFFF; // jeśli tak, to zwracamy wartość 0xFFFFFFFF, co oznacza błąd dekodowa-  
nia  
  
    if (pulses[0] < NEC_AGC_MARK_MIN_US || pulses[0] > NEC_AGC_MARK_MAX_US) // sprawdzamy, czy  
    pierwszy impuls jest w zakresie AGC  
        return 0xFFFFFFFF;  
    if (pulses[1] < NEC_AGC_SPACE_MIN_US || pulses[1] > NEC_AGC_SPACE_MAX_US) // sprawdzamy, czy  
    drugi impuls jest w zakresie AGC  
        return 0xFFFFFFFF; // jeśli nie, to zwracamy wartość 0xFFFFFFFF, co oznacza błąd dekodowa-  
nia  
  
    uint32_t decoded_code = 0; // zmienna do przechowywania zdekodowanego kodu IR, inicjalizowana  
na 0  
    for (int i = 0; i < 32; i++) // iterujemy przez 32 bity danych w ramce IR NEC  
    {  
        uint32_t bit_mark = pulses[2 + (2 * i)]; // pobieramy czas trwania impulsu bitu, który jest  
    przechowywany w buforze impulsów  
        uint32_t bit_space = pulses[2 + (2 * i) + 1]; // pobieramy czas trwania przerwy bitu, który  
    jest przechowywany w buforze impulsów  
  
        if (bit_mark < NEC_BIT_MARK_MIN_US || bit_mark > NEC_BIT_MARK_MAX_US) // sprawdzamy, czy  
    impuls bitu jest w zakresie  
        return 0xFFFFFFFF; // jeśli nie, to zwracamy wartość 0xFFFFFFFF, co oznacza błąd deko-  
dowania  
  
        decoded_code <= 1; // przesuwamy zdekodowany kod o 1 bit w lewo, aby przygotować miejsce  
na nowy bit  
        if (bit_space >= NEC_BIT_1_SPACE_MIN_US && bit_space <= NEC_BIT_1_SPACE_MAX_US) // spraw-  
dzamy, czy przerwa bitu jest w zakresie dla bitu 1  
        {  
            decoded_code |= 1; // jeśli tak, to ustawiamy najmłodszy bit zdekodowanego kodu na 1  
        }  
        else if (bit_space >= NEC_BIT_0_SPACE_MIN_US && bit_space <= NEC_BIT_0_SPACE_MAX_US) // /  
sprawdzamy, czy przerwa bitu jest w zakresie dla bitu 0
```



Adrian Popielarczyk 21295, Zuzanna Orzechowska 21284

```
{  
}  
else  
{  
    return 0xFFFFFFFF; // jeśli przerwa bitu nie jest w zakresie dla bitu 0 ani bitu 1, to  
// zwracamy wartość 0xFFFFFFFF, co oznacza błąd dekodowania  
}  
}  
return decoded_code; // zwracamy zdekodowany kod IR, który jest 32-bitową liczbą, gdzie 16 bi-  
tów to adres, a 16 bitów to dane  
/* USER CODE END 0 */
```

Kod w sekcji USER CODE 1: (rozpoczęcie funkcji main):

```
/* USER CODE BEGIN 1 */  
char lcd_buf[17]; // bufor do przechowywania tekstu do wyświetlenia na LCD, 16 znaków + 1 na  
znak końca  
char lcd_action_msg[17]; // bufor do przechowywania komunikatu akcji do wyświetlenia na LCD, 16  
znaków + 1 na znak końca  
char clear_line[17]; // bufor do przechowywania pustej linii do wyświetlenia na LCD, 16 znaków  
+ 1 na znak końca  
memset(clear_line, ' ', 16); // wypełniamy bufor pustymi znakami, aby wyczyścić linię na LCD  
clear_line[16] = '\0'; // dodajemy znak końca łańcucha, aby bufor był poprawnie zakończony  
/* USER CODE END 1 */
```

Kod w sekcji USER CODE 2:

```
/* USER CODE BEGIN 2 */  
LCD_Init(); // inicjalizacja LCD  
LCD_DrawFace(1); //rysowanie buźki na LCD  
HAL_Delay(1000); //opóźnienie  
LCD_DrawFace(0); // rysowanie buźki 2  
HAL_Delay(1000);  
LCD_DrawFace(2);  
HAL_Delay(1000);  
play_melody(happy_bounce); // odtwarzanie melodii "Happy Bounce"  
lcd_initialized = true; // ustawiamy flagę inicjalizacji LCD na true  
LCD_SetCursor(0, 0); // ustawiamy kursor na początek pierwszej linii LCD  
LCD_SendString("PROJEKT SWIM"); // wysyłamy tekst do LCD  
LCD_SetCursor(1, 0); // ustawiamy kursor na początek drugiej linii LCD  
LCD_SendString("Czekam na B1..."); // wysyłamy tekst do LCD, informujący o oczekiwaniu na przy-  
cisk B1  
  
HAL_TIM_Base_Start(&htim3); // uruchamiamy timer 3, który będzie używany do przechwytywania im-  
pulsów IR  
__HAL_TIM_SET_COUNTER(&htim3, 0); // ustawiamy licznik timera 3 na 0, aby rozpocząć liczenie od  
zera  
last_edge_time = __HAL_TIM_GET_COUNTER(&htim3); // zapisujemy aktualny czas jako czas ostat-  
niego zbocza
```



Raport z budowy robota

Adrian Popielarczyk 21295, Zuzanna Orzechowska 21284

```
pulse_index = 0; // inicjalizujemy indeks bufora impulsów na 0
ir_frame_ready = false; // inicjalizujemy flagęgotowości ramki IR na false
ir_repeat_detected = false; // inicjalizujemy flagę wykrycia powtarzającej ramki IR na false
capture_started = false; // inicjalizujemy flagę przechwytywania impulsów IR na false

while (HAL_GPIO_ReadPin(B1_GPIO_Port, B1_Pin) == GPIO_PIN_SET) // czekamy, aż przycisk B1 zostanie naciśnięty
{
    HAL_Delay(100);
    HAL_GPIO_TogglePin(LED_GPIO_Port, LED_Pin); // migamy diodą LED, aby pokazać, że czekamy na naciśnięcie przycisku
}
HAL_GPIO_WritePin(LED_GPIO_Port, LED_Pin, GPIO_PIN_RESET); // wyłączaemy diodę LED po naciśnięciu przycisku B1

LCD_Clear(); // czyścimy LCD
LCD_SetCursor(0, 0); // ustawiamy kursor na początek pierwszej linii LCD
LCD_SendString("Tryb: Auto"); // wysyłamy tekst do LCD, informujący o trybie automatycznym
LCD_SetCursor(1, 0); // ustawiamy kursor na początek drugiej linii LCD
LCD_SendString("Start!"); // wysyłamy tekst do LCD, informujący o rozpoczęciu działania
HAL_Delay(1000); // czekamy 1 sekundę, aby użytkownik mógł zobaczyć komunikat na LCD
LCD_DrawFace(1); // rysujemy buźkę na LCD, aby pokazać, że robot jest gotowy do działania

HAL_TIM_PWM_Start(&htim2, TIM_CHANNEL_1); // uruchamiamy PWM na kanale 1 timera 2, który steruje lewym silnikiem
HAL_TIM_PWM_Start(&htim2, TIM_CHANNEL_2); // uruchamiamy PWM na kanale 2 timera 2, który steruje prawym silnikiem
stop();
/* USER CODE END 2 */
```

Kod w sekcji USER CODE WHILE:

```
/* USER CODE BEGIN WHILE */
while (1)
{
    uint32_t ir_code_to_process = 0xFFFFFFFF; // zmienna do przechowywania kodu IR do przetwarzania, inicjalizowana na 0xFFFFFFFF, co oznacza brak kodu do przetworzenia
    bool is_repeat_frame = false; // flaga informująca, czy ramka IR jest powtarzająca, inicjalizowana na false

    if (ir_frame_ready) // jeśli ramka IR jest gotowa do przetworzenia
    {
        uint32_t local_pulse_widths[IR_RX_BUFFER_SIZE]; // bufor do przechowywania impulsów IR do przetworzenia
        uint8_t local_pulse_count; // zmienna do przechowywania liczby impulsów IR do przetwarzania, inicjalizowana na 0

        HAL_NVIC_DisableIRQ EXTI15_10_IRQn); // wyłączaemy przerwania zewnętrzne, aby uniknąć zakłóceń podczas przetwarzania ramki IR
```



Raport z budowy robota

Adrian Popielarczyk 21295, Zuzanna Orzechowska 21284

```
local_pulse_count = pulse_index; // zapisujemy liczbę impulsów IR do przetworzenia,
która jest aktualnym indeksem bufora impulsów
memcpy(local_pulse_widths, (void *)pulse_widths, local_pulse_count * sizeof(uint32_t));
// kopiujemy impulsów IR do lokalnego bufora, aby uniknąć modyfikacji oryginalnego bufora podczas
przetwarzania

pulse_index = 0; // resetujemy indeks bufora impulsów, aby przygotować go do przechwytywania kolejnej ramki IR
ir_frame_ready = false; // resetujemy flagę gotowości ramki IR, aby uniknąć ponownego
przetwarzania tej samej ramki
HAL_NVIC_EnableIRQ(EXTI15_10_IRQn); // włączamy przerwania zewnętrzne, aby umożliwić
dalejsze przechwytywanie impulsów IR

uint32_t decoded_code = decode_nec(local_pulse_widths, local_pulse_count); // dekodujemy
ramkę IR NEC, przekazując lokalny bufor impulsów i liczbę impulsów do funkcji dekodującej

if (decoded_code != 0xFFFFFFFF) // jeśli dekodowanie ramki IR zakończyło się sukcesem,
czyli zwrócono poprawny kod IR
{
    ir_code_to_process = decoded_code; // zapisujemy zdekodowany kod IR do zmiennej do
przetwarzania
    last_good_ir_code = decoded_code; // zapisujemy ostatni poprawny kod IR, który został
przetworzony
    is_repeat_frame = false; // ustawiamy flagę powtarzającej ramki IR na false, ponieważ
właśnie przetworzono nową ramkę IR
}
else // jeśli dekodowanie ramki IR zakończyło się niepowodzeniem, czyli zwrócono wartość
0xFFFFFFFF
{
    LCD_SetCursor(0, 0); // ustawiamy kurSOR na początek pierwszej linii LCD
    LCD_SendString(clear_line); // czyścimy pierwszą linię LCD
    LCD_SetCursor(1, 0); // ustawiamy kurSOR na początek drugiej linii LCD
    LCD_SendString(clear_line); // czyścimy drugą linię LCD
    LCD_SetCursor(0, 0); // ustawiamy kurSOR na początek pierwszej linii LCD
    LCD_SendString("Bład dekodowania"); // wysyłamy komunikat o błędzie dekodowania
ramki IR na LCD
}
else if (ir_repeat_detected) // jeśli wykryto powtarzającą ramkę IR
{
    HAL_NVIC_DisableIRQ(EXTI15_10_IRQn); // wyłączamy przerwania zewnętrzne, aby uniknąć
zakłóceń podczas przetwarzania powtarzającej ramki IR
    ir_repeat_detected = false; // resetujemy flagę wykrycia powtarzającej ramki IR, aby
uniknąć ponownego przetwarzania tej samej ramki
    HAL_NVIC_EnableIRQ(EXTI15_10_IRQn); // włączamy przerwania zewnętrzne, aby umożliwić
dalejsze przechwytywanie impulsów IR

    if (last_good_ir_code != 0xFFFFFFFF) // jeśli ostatni poprawny kod IR jest różny od
0xFFFFFFFF, co oznacza, że wcześniej przetworzono poprawną ramkę IR
```



Raport z budowy robota

Adrian Popielarczyk 21295, Zuzanna Orzechowska 21284

```
{  
    ir_code_to_process = last_good_ir_code; // zapisujemy ostatni poprawny kod IR do  
zmiennej do przetworzenia  
    is_repeat_frame = true; // ustawiamy flagę powtarzającej ramki IR na true, ponieważ  
właśnie przetworzono powtarzającą ramkę IR  
}  
else  
{  
}  
}  
  
if (ir_code_to_process != 0xFFFFFFFF) // jeśli jest kod IR do przetworzenia, czyli zdekodo-  
wano poprawną ramkę IR lub wykryto powtarzającą ramkę IR  
{  
    bool allow_processing = true; // flaga informująca, czy przetwarzanie kodu IR jest do-  
zwolone, inicjalizowana na true  
    memset(lcd_action_msg, 0, sizeof(lcd_action_msg)); // czyścimy bufor komunikatu akcji,  
aby przygotować go do nowego komunikatu  
  
    // wyświetlamy kod IR na LCD  
    LCD_SetCursor(0, 0);  
    LCD_SendString(clear_line);  
    LCD_SetCursor(1, 0);  
    LCD_SendString(clear_line);  
    LCD_SetCursor(0, 0);  
  
    if (is_repeat_frame) // jeśli jest to powtarzająca ramka IR  
{  
        sprintf(lcd_buf, "IR RPT(0x%04X)", (unsigned int)(ir_code_to_process & 0xFFFF)); //  
wyświetlamy kod IR jako powtarzający, ograniczając go do 16 bitów  
    }  
    else  
{  
        sprintf(lcd_buf, "IR:0x%08X", (unsigned int)ir_code_to_process); // wyświetlamy kod  
IR jako pełny 32-bitowy kod  
    }  
    LCD_SendString(lcd_buf); // wysyłamy kod IR do LCD  
  
    bool is_discrete_event_code = // sprawdzamy, czy kod IR jest jednym z dyskretnych kodów  
zdarzeń, które mają specjalne znaczenie  
    (ir_code_to_process == 0x00FF6897) || // kod IR dla trybu zdalnego  
    (ir_code_to_process == 0x00FFB04F) || // kod IR dla trybu automatycznego  
    (ir_code_to_process == 0x00FF38C7) || // kod IR dla komendy STOP  
    (ir_code_to_process == 0x00FFA25D) || // kod IR dla prędkości wolnej  
    (ir_code_to_process == 0x00FFE21D); // kod IR dla prędkości szybkiej  
  
    if (is_discrete_event_code) // jeśli kod IR jest jednym z dyskretnych kodów zdarzeń  
{
```



Raport z budowy robota

Adrian Popielarczyk 21295, Zuzanna Orzechowska 21284

```
if (ir_code_to_process == last_processed_discrete_ir_code && // sprawdzamy, czy kod
IR jest taki sam jak ostatnio przetworzony dyskretny kod
    (HAL_GetTick() - last_processed_discrete_ir_time < DISCRETE_CMD_MIN_INTE-
RVAL_MS)) // i czy czas od ostatniego przetworzenia jest mniejszy niż minimalny interwał między
dyskretnymi komendami
{
    allow_processing = false; // jeśli tak, to ustawiamy flagę dozwolenia przetwa-
rzania na false, aby zignorować powtarzające się dyskretne komendy
    if (is_repeat_frame) // jeśli jest to powtarzająca ramka IR
        sprintf(lcd_action_msg, "Hold Ignored"); // wyświetlamy komunikat o igno-
rowaniu powtarzającej się komendy
    else
        sprintf(lcd_action_msg, "Debounce Ignored"); // wyświetlamy komunikat o
ignorowaniu powtarzającej się komendy z powodu debouncingu
    }
}

if (allow_processing) // jeśli przetwarzanie kodu IR jest dozwolone
{
    bool discrete_action_has_been_processed_this_cycle = false; // flaga informująca,
czy przetworzono dyskretną akcję w tej iteracji, inicjalizowana na false

    if (ir_code_to_process == 0x00FF6897) // kod IR dla trybu zdalnego
    {
        if (control_mode != 1) // jeśli aktualny tryb nie jest trybem zdalnym
        {
            control_mode = 1; // ustawiamy tryb na zdalny
            stop(); // zatrzymujemy robota
            is_moving_remotely = false; // ustawiamy flagę ruchu zdalnego na false, po-
nieważ nie ma jeszcze żadnej komendy ruchu
        }
        sprintf(lcd_action_msg, "Tryb: Zdalny"); // wyświetlamy komunikat o przejściu
do trybu zdalnego
        discrete_action_has_been_processed_this_cycle = true; // ustawiamy flagę prze-
tworzenia dyskretnej akcji na true, ponieważ przetworzono zmianę trybu
    }
    else if (ir_code_to_process == 0x00FFB04F) // kod IR dla trybu automatycznego
    {
        if (control_mode != 0) // jeśli aktualny tryb nie jest trybem automatycznym
        {
            control_mode = 0; // ustawiamy tryb na automatyczny
            stop(); // zatrzymujemy robota
            is_moving_remotely = false; // ustawiamy flagę ruchu zdalnego na false, po-
nieważ nie ma jeszcze żadnej komendy ruchu
            bitSkretu = 0; // resetujemy kierunek skrętu, ponieważ w trybie automatycz-
nym robot będzie szukał linii
        }
        sprintf(lcd_action_msg, "Tryb: Auto"); // wyświetlamy komunikat o przejściu do
trybu automatycznego
    }
}
```



Adrian Popielarczyk 21295, Zuzanna Orzechowska 21284

```
discrete_action_has_been_processed_this_cycle = true; // ustawiamy flagę przetworzenia dyskretnej akcji na true, ponieważ przetworzono zmianę trybu
}
else if (control_mode == 1) // jeśli aktualny tryb to tryb zdalny
{
    switch (ir_code_to_process) // sprawdzamy, jaki kod IR został przetworzony i wykonujemy odpowiednią akcję
    {
        case 0x00FF18E7: // kod IR dla przycisku "Naprzód"
            dzidaDoPrzodu(); // funkcja do jazdy do przodu
            sprintf(lcd_action_msg, "Cmd: Naprzod"); // wyświetlamy komunikat o komendzie jazdy do przodu
            is_moving_remotely = true; // ustawiamy flagę ruchu zdalnego na true, ponieważ robot będzie się poruszał
            last_remote_movement_command_time = HAL_GetTick(); // zapisujemy czas ostatniej komendy ruchu zdalnego
            break;
        case 0x00FF4AB5: // kod IR dla przycisku "Wstecz"
            doTylu(); // funkcja do jazdy do tyłu
            sprintf(lcd_action_msg, "Cmd: Tyl"); // wyświetlamy komunikat o komendzie jazdy do tyłu
            is_moving_remotely = true; // ustawiamy flagę ruchu zdalnego na true, ponieważ robot będzie się poruszał
            last_remote_movement_command_time = HAL_GetTick(); // zapisujemy czas ostatniej komendy ruchu zdalnego
            break;
        case 0x00FF10EF: // kod IR dla przycisku "Obrót w lewo"
            lewy90(); // funkcja do obrotu w lewo
            sprintf(lcd_action_msg, "Cmd: Obr Lewo"); // wyświetlamy komunikat o komendzie obrotu w lewo
            is_moving_remotely = true; // ustawiamy flagę ruchu zdalnego na true, ponieważ robot będzie się poruszał
            last_remote_movement_command_time = HAL_GetTick(); // zapisujemy czas ostatniej komendy ruchu zdalnego
            break;
        case 0x00FF5AA5: // kod IR dla przycisku "Obrót w prawo"
            prawy90(); // funkcja do obrotu w prawo
            sprintf(lcd_action_msg, "Cmd: Obr Prawo"); // wyświetlamy komunikat o komendzie obrotu w prawo
            is_moving_remotely = true; // ustawiamy flagę ruchu zdalnego na true, ponieważ robot będzie się poruszał
            last_remote_movement_command_time = HAL_GetTick(); // zapisujemy czas ostatniej komendy ruchu zdalnego
            break;
        case 0x00FF22DD: // kod IR dla przycisku "Skręt w lewo"
            skretWLewo(); // funkcja do skrętu w lewo
            sprintf(lcd_action_msg, "Cmd: Skret L"); // wyświetlamy komunikat o komendzie skrętu w lewo
    }
}
```



Adrian Popielarczyk 21295, Zuzanna Orzechowska 21284

```
is_moving_remotely = true; // ustawiamy flagę ruchu zdalnego na true, ponieważ robot będzie się poruszał
last_remote_movement_command_time = HAL_GetTick(); // zapisujemy czas ostatniej komendy ruchu zdalnego
break;
case 0x00FFC23D: // kod IR dla przycisku "Skręt w prawo"
    skretWPrawo(); // funkcja do skrętu w prawo
    sprintf(lcd_action_msg, "Cmd: Skret P"); // wyświetlamy komunikat o komendzie skrętu w prawo
    is_moving_remotely = true; // ustawiamy flagę ruchu zdalnego na true, ponieważ robot będzie się poruszał
    last_remote_movement_command_time = HAL_GetTick(); // zapisujemy czas ostatniej komendy ruchu zdalnego
    break;

case 0x00FF38C7: // kod IR dla przycisku "STOP"
    stop();
    sprintf(lcd_action_msg, "Cmd: STOP"); // wyświetlamy komunikat o komendzie zatrzymania
    is_moving_remotely = false; // ustawiamy flagę ruchu zdalnego na false, ponieważ robot został zatrzymany
    discrete_action_has_been_processed_this_cycle = true; // ustawiamy flagę przetworzenia dyskretnej akcji na true, ponieważ przetworzono komendę zatrzymania
    break;
case 0x00FFA25D: // kod IR dla przycisku "Prędkość wolna"
    duty = DUTY_SLOW; // ustawiamy prędkość na wolną
    sprintf(lcd_action_msg, "Predk: Wolno"); // wyświetlamy komunikat o zmianie prędkości na wolną
    if (is_moving_remotely) // jeśli robot jest w ruchu zdalnym
        last_remote_movement_command_time = HAL_GetTick(); // zapisujemy czas ostatniej komendy ruchu zdalnego
    discrete_action_has_been_processed_this_cycle = true; // ustawiamy flagę przetworzenia dyskretnej akcji na true, ponieważ przetworzono zmianę prędkości
    break;
case 0x00FFE21D: // kod IR dla przycisku "Prędkość szybka"
    duty = DUTY_FAST; // ustawiamy prędkość na szybką
    sprintf(lcd_action_msg, "Predk: Szybko"); // wyświetlamy komunikat o zmianie prędkości na szybką
    if (is_moving_remotely) // jeśli robot jest w ruchu zdalnym
        last_remote_movement_command_time = HAL_GetTick(); // zapisujemy czas ostatniej komendy ruchu zdalnego
    discrete_action_has_been_processed_this_cycle = true; // ustawiamy flagę przetworzenia dyskretnej akcji na true, ponieważ przetworzono zmianę prędkości
    break;

default:
    if (lcd_action_msg[0] == '\0') // jeśli bufor komunikatu akcji jest pusty, czyli nie przetworzono jeszcze żadnej dyskretnej akcji
    {
```



Adrian Popielarczyk 21295, Zuzanna Orzechowska 21284

```
sprintf(lcd_action_msg, "IR Kod ???"); // wyświetlamy komunikat o nie-
znanym kodzie IR
    }
    break;
}
}
else
{
    if (lcd_action_msg[0] == '\0' && // kod IR nie jest jednym z dyskretnych kodów
zdarzeń, które mają specjalne znaczenie
        ir_code_to_process != 0x00FF6897 && // kod IR nie jest kodem dla trybu
zdalnego
        ir_code_to_process != 0x00FFB04F) // kod IR nie jest kodem dla trybu auto-
matycznego
    {
        sprintf(lcd_action_msg, "IR (Tryb Auto)"); // wyświetlamy komunikat o prze-
twarzaniu kodu IR w trybie automatycznym
    }
}

if (discrete_action_has_been_processed_this_cycle) // jeśli przetworzono dyskretną
akcję w tej iteracji
{
    last_processed_discrete_ir_code = ir_code_to_process; // zapisujemy kod IR jako
ostatnio przetworzony dyskretny kod
    last_processed_discrete_ir_time = HAL_GetTick(); // zapisujemy czas ostatniego
przetworzenia dyskretnej akcji
}

if (lcd_action_msg[0] != '\0') // jeśli bufor komunikatu akcji nie jest pusty, czyli
przetworzono jakąś akcję
{
    LCD_SetCursor(1, 0); // ustawiamy kursor na początek drugiej linii LCD
    LCD_SendString(lcd_action_msg); // wysyłamy komunikat akcji do LCD
}

if (control_mode == 1) // jeśli aktualny tryb to tryb zdalny
{
    if (is_moving_remotely && (HAL_GetTick() - last_remote_movement_command_time > RE-
MOTE_MOVEMENT_TIMEOUT)) // jeśli robot jest w ruchu zdalnym i minął czas oczekiwania na kolejną ko-
mendę ruchu zdalnego
    {
        stop(); // zatrzymujemy robota
        is_moving_remotely = false; // ustawiamy flagę ruchu zdalnego na false, ponieważ
robot został zatrzymany

        // wyświetlamy komunikat o przekroczeniu limitu czasu dla ruchu zdalnego
    }
}
```



Adrian Popielarczyk 21295, Zuzanna Orzechowska 21284

```
LCD_SetCursor(0, 0);
LCD_SendString(clear_line);
LCD_SetCursor(1, 0);
LCD_SendString(clear_line);
LCD_SetCursor(0, 0);
LCD_SendString("Pilot Timeout!");
LCD_SetCursor(1, 0);
LCD_SendString("STOP");
}

HAL_Delay(20);
}
else
{
    lewy = HAL_GPIO_ReadPin(IR1_GPIO_Port, IR1_Pin); // odczytujemy stan lewego czujnika IR
    srodek = HAL_GPIO_ReadPin(IR2_GPIO_Port, IR2_Pin); // odczytujemy stan środkowego czuj-
nik IR
    prawy = HAL_GPIO_ReadPin(IR3_GPIO_Port, IR3_Pin); // odczytujemy stan prawego czujnika
IR

    //poniszy kod dla poruszania się automatycznego po wykryciu czarnej linii został opisany w poprzednim kodzie do Milestone II
if (lewy && srodek && prawy) {
    duty = DUTY_SLOW;
    dzidaDoPrzodu();
}
else if (lewy && srodek && !prawy) {
    duty = DUTY_SLOW;
    skretWLewo();
    bitSkretu=1;
}
else if (prawy && srodek && !lewy) {
    duty = DUTY_SLOW;
    skretWPravo();
    bitSkretu=-1;
}
else if (srodek && !lewy && !prawy) {
    duty = DUTY_SLOW;
    dzidaDoPrzodu();
}
else if (lewy && !srodek && !prawy) {
    duty = DUTY_SLOW;
    skretWLewo();
    bitSkretu=1;
}
else if (prawy && !srodek && !lewy) {
    duty = DUTY_SLOW;
    skretWPravo();
    bitSkretu=-1;
}
```



Adrian Popielarczyk 21295, Zuzanna Orzechowska 21284

```
        else {
            duty = DUTY_SLOW;
            searchLine(bitSkretu);
        }

        HAL_Delay(50);
    }
/* USER CODE END WHILE */
```

Poniżej znajdują się fragmenty kodu do pliku z kodem obsługującym wyświetlacz lcd – ldchd44780.c:

```
#include "lcd_hd44780.h"
#include "stm32f4xx_hal.h"
#include "string.h"
#include <stdint.h>

static void LCD_EnablePulse(void) { // funkcja wysyłająca impuls do linii EN
    HAL_GPIO_WritePin(LCD_EN_GPIO_Port, LCD_EN_Pin, GPIO_PIN_SET); // ustawienie linii EN na wysoki stan
    HAL_Delay(1);
    HAL_GPIO_WritePin(LCD_EN_GPIO_Port, LCD_EN_Pin, GPIO_PIN_RESET); // ustawienie linii EN na niski stan
    HAL_Delay(1);
}

static void LCD_Write8Bits(uint8_t data) { // funkcja wysyłająca 8 bitów danych do LCD
    HAL_GPIO_WritePin(LCD_D0_GPIO_Port, LCD_D0_Pin, (data & 0x01) ? GPIO_PIN_SET : GPIO_PIN_RESET);
    // ustawienie linii D0 na wysoki lub niski stan w zależności od bitu
    HAL_GPIO_WritePin(LCD_D1_GPIO_Port, LCD_D1_Pin, (data & 0x02) ? GPIO_PIN_SET : GPIO_PIN_RESET);
    // ustawienie linii D1 na wysoki lub niski stan w zależności od bitu
    HAL_GPIO_WritePin(LCD_D2_GPIO_Port, LCD_D2_Pin, (data & 0x04) ? GPIO_PIN_SET : GPIO_PIN_RESET);
    // ustawienie linii D2 na wysoki lub niski stan w zależności od bitu
    HAL_GPIO_WritePin(LCD_D3_GPIO_Port, LCD_D3_Pin, (data & 0x08) ? GPIO_PIN_SET : GPIO_PIN_RESET);
    // ustawienie linii D3 na wysoki lub niski stan w zależności od bitu
    HAL_GPIO_WritePin(LCD_D4_GPIO_Port, LCD_D4_Pin, (data & 0x10) ? GPIO_PIN_SET : GPIO_PIN_RESET);
    // ustawienie linii D4 na wysoki lub niski stan w zależności od bitu
    HAL_GPIO_WritePin(LCD_D5_GPIO_Port, LCD_D5_Pin, (data & 0x20) ? GPIO_PIN_SET : GPIO_PIN_RESET);
    // ustawienie linii D5 na wysoki lub niski stan w zależności od bitu
    HAL_GPIO_WritePin(LCD_D6_GPIO_Port, LCD_D6_Pin, (data & 0x40) ? GPIO_PIN_SET : GPIO_PIN_RESET);
    // ustawienie linii D6 na wysoki lub niski stan w zależności od bitu
    HAL_GPIO_WritePin(LCD_D7_GPIO_Port, LCD_D7_Pin, (data & 0x80) ? GPIO_PIN_SET : GPIO_PIN_RESET);
    // ustawienie linii D7 na wysoki lub niski stan w zależności od bitu
    LCD_EnablePulse(); // wywołanie funkcji wysyłającej impuls do linii EN
}

static void LCD_Send(uint8_t data, uint8_t rs) { // funkcja wysyłająca dane lub komendę do LCD
    HAL_GPIO_WritePin(LCD_RS_GPIO_Port, LCD_RS_Pin, rs ? GPIO_PIN_SET : GPIO_PIN_RESET); // ustawienie linii RS na wysoki lub niski stan w zależności od tego, czy wysyłamy dane (1) czy komendę (0)
```



Adrian Popielarczyk 21295, Zuzanna Orzechowska 21284

```
LCD_Write8Bits(data); // wysłanie 8 bitów danych lub komendy do LCD
}

void LCD_SendCommand(uint8_t cmd) { // funkcja wysyłająca komendę do LCD
    LCD_Send(cmd, 0); // wywołanie funkcji wysyłającej komendę z parametrem rs ustawionym na 0 (komenda)
}

void LCD_SendData(uint8_t data) { // funkcja wysyłająca dane do LCD
    LCD_Send(data, 1); // wywołanie funkcji wysyłającej dane z parametrem rs ustawionym na 1 (dane)
}

void LCD_Clear(void) { // funkcja czyszcząca ekran LCD
    LCD_SendCommand(0x01); // wysłanie komendy czyszczenia ekranu
    HAL_Delay(2);
}

void LCD_SetCursor(uint8_t row, uint8_t col) { // funkcja ustawiająca kursor na określonej pozycji
    uint8_t addr = (row == 0) ? 0x00 : 0x40; // adres początkowy dla pierwszego wiersza to 0x00, a dla drugiego to 0x40
    LCD_SendCommand(0x80 | (addr + col)); // wysłanie komendy ustawiającej kursor na odpowiedniej pozycji
}

void LCD_SendString(char* str) { // funkcja wysyłająca łańcuch znaków do LCD
    while (*str) { // dopóki nie napotkamy znaku końca łańcucha
        LCD_SendData(*str++); // wysyłamy aktualny znak i przechodzimy do następnego
    }
}

void LCD_Init(void) { // funkcja inicjalizująca LCD
    HAL_Delay(100);

    HAL_GPIO_WritePin(LCD_RS_GPIO_Port, LCD_RS_Pin, GPIO_PIN_RESET); // ustawienie linii RS na niski stan (komenda)
    HAL_GPIO_WritePin(LCD_EN_GPIO_Port, LCD_EN_Pin, GPIO_PIN_RESET); // ustawienie linii EN na niski stan (wyłączone)
    LCD_Write8Bits(0x30); // wysłanie pierwszej komendy inicjalizacyjnej
    HAL_Delay(5);
    LCD_Write8Bits(0x30); // wysłanie drugiej komendy inicjalizacyjnej
    HAL_Delay(5);
    LCD_Write8Bits(0x30); // wysłanie trzeciej komendy inicjalizacyjnej
    HAL_Delay(5);
    LCD_SendCommand(0x38); // ustawienie trybu 8-bitowego, 2 linie, 5x8 czcionka
    HAL_Delay(5);
    LCD_SendCommand(0x0F); // włączenie wyświetlacza, kurSORA i migania kurSORa
    HAL_Delay(2);
    LCD_SendCommand(0x01); // czyszczenie ekranu
    HAL_Delay(2);
}
```



Adrian Popielarczyk 21295, Zuzanna Orzechowska 21284

```
LCD_SendCommand(0x02); // ustawienie kursora na początek
HAL_Delay(2);
LCD_Clear();
}

//easter egg

void LCD_CreateChar(uint8_t location, uint8_t charmap[]) { // funkcja tworząca niestandardowy znak
w pamięci LCD
    location &= 0x7; // tylko 0-7
    LCD_SendCommand(0x40 | (location << 3)); // ustawienie adresu pamięci dla niestandardowego
znaku
    for (int i = 0; i < 8; i++) { // wysyłanie 8 bajtów danych dla niestandardowego znaku
        LCD_SendData(charmap[i]); // wysyłanie każdego bajtu z tablicy charmap
    }
}

// Lewo
uint8_t eye_left[8] = { // tablica z danymi dla oczu skierowanych w lewo, wartości w tablicy to
bity reprezentujące piksele
    0b11111,
    0b10001,
    0b10101,
    0b10101,
    0b10001,
    0b11111,
    0b00000,
    0b00000
};

// Prosto
uint8_t eye_forward[8] = { // tablica z danymi dla oczu skierowanych prosto
    0b11111,
    0b10001,
    0b10001,
    0b10101,
    0b10001,
    0b11111,
    0b00000,
    0b00000
};

// Prawo
uint8_t eye_right[8] = { // tablica z danymi dla oczu skierowanych w prawo
    0b11111,
    0b10001,
    0b10101,
    0b10101,
```



Adrian Popielarczyk 21295, Zuzanna Orzechowska 21284

```
0b10001,  
0b11111,  
0b00000,  
0b00000  
};  
  
// Nos – prosto  
uint8_t nose_forward[8] = { // tablica z danymi dla nosa skierowanego prosto  
    0b00000,  
    0b00000,  
    0b00100,  
    0b00100,  
    0b01010,  
    0b01010,  
    0b10001,  
    0b00000  
};  
  
// Nos – w lewo  
uint8_t nose_left[8] = { // tablica z danymi dla nosa skierowanego w lewo  
    0b00000,  
    0b00000,  
    0b00010,  
    0b00010,  
    0b00101,  
    0b00101,  
    0b01001,  
    0b00000  
};  
  
// Nos – w prawo  
uint8_t nose_right[8] = { // tablica z danymi dla nosa skierowanego w prawo  
    0b00000,  
    0b00000,  
    0b01000,  
    0b01000,  
    0b10100,  
    0b10100,  
    0b10010,  
    0b00000  
};  
  
void LCD_DrawFace(uint8_t direction) { // funkcja rysująca twarz na LCD  
    // direction: 0 = lewo, 1 = prosto, 2 = prawo  
  
    switch (direction) { // sprawdzenie kierunku patrzenia  
        case 0: // lewo
```



Adrian Popielarczyk 21295, Zuzanna Orzechowska 21284

```
LCD_CreateChar(0, eye_left); // tworzenie niestandardowego znaku dla oczu skierowanych
w lewo
LCD_CreateChar(1, nose_left); // tworzenie niestandardowego znaku dla nosa skierowanego
w lewo
break;
case 1: // prosto
LCD_CreateChar(0, eye_forward); // tworzenie niestandardowego znaku dla oczu skierowanych prosto
LCD_CreateChar(1, nose_forward); // tworzenie niestandardowego znaku dla nosa skierowanego prosto
break;
case 2: // prawo
LCD_CreateChar(0, eye_right); // tworzenie niestandardowego znaku dla oczu skierowanych
w prawo
LCD_CreateChar(1, nose_right); // tworzenie niestandardowego znaku dla nosa skierowanego w prawo
break;
}

LCD_Clear();

// Góra linia: oczy
LCD_SetCursor(0, 6); // ustawienie kurSORA na początek górnej linii
LCD_SendData(0); // wysłanie niestandardowego znaku dla oczu
LCD_SendData(' '); // wysłanie spacji
LCD_SendData(0); // wysłanie kolejnego niestandardowego znaku dla oczu

// Dolna linia: nos
LCD_SetCursor(1, 7); // ustawienie kurSORa na początek dolnej linii
LCD_SendData(1); // wysłanie niestandardowego znaku dla nosa
}
```

Poniżej przedstawiono fragment kodu odpowiedzialnego za przerwania w pliku stm32f4xx_it.c

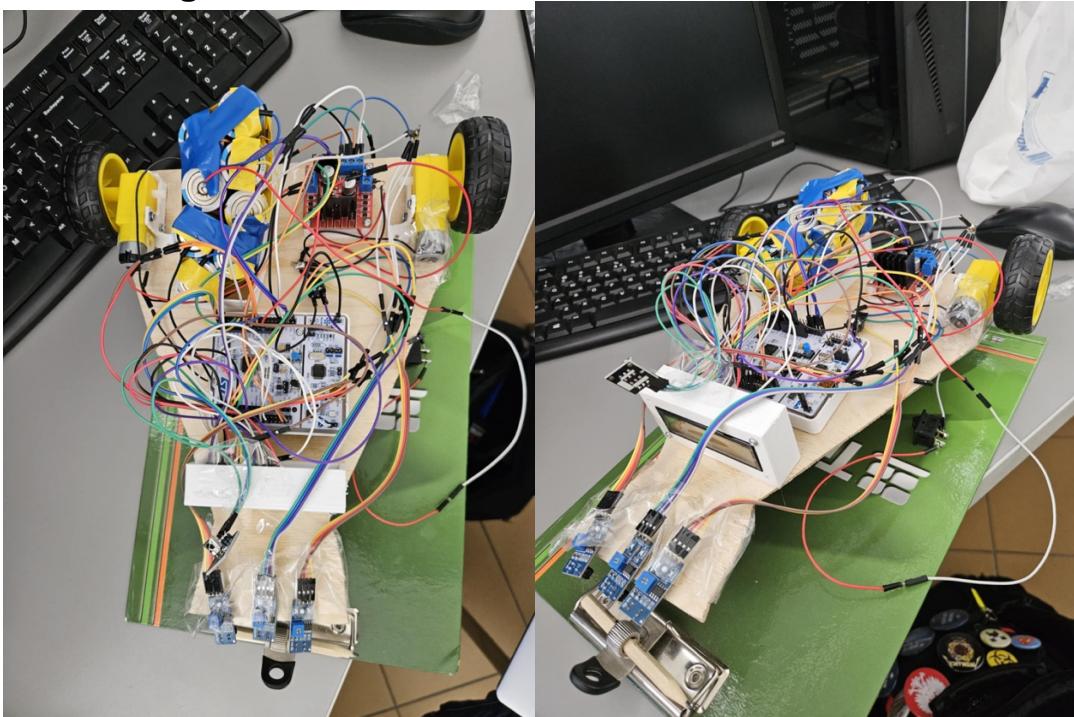
```
void EXTI15_10_IRQHandler(void) //funkcja odpowiedzialna za przerwanie z GPIO pinów 10-15, potrzebna do obsługi modułu podczerwieni
{
    /* USER CODE BEGIN EXTI15_10_IRQHandler_0 */

    /* USER CODE END EXTI15_10_IRQHandler_0 */
    HAL_GPIO_EXTI_IRQHandler(IR_Pin); //wywołanie funkcji obsługującej przerwanie z GPIO, w tym przypadku z pinu IR
    /* USER CODE BEGIN EXTI15_10_IRQHandler_1 */

    /* USER CODE END EXTI15_10_IRQHandler_1 */
}
```

Adrian Popielarczyk 21295, Zuzanna Orzechowska 21284

Zdjęcia opracowanego robota



Wnioski:

Robot poprawnie wykonuje trasę – zarówno automatyczną po linii jak i zdalnie sterowaną za pomocą pilota i modułu podczerwieni. Dodatkowo wyświetlacz LCD działa bez zarzutu, podczas automatycznej jazdy na ekranie pokazuje się buzia z oczami i nosem, ułożenie oczu na ekranie zmienia się wraz ze zmianą kierunku jazdy robota (robot patrzy, w tą stronę, w którą jedzie). Kolejną funkcjonalnością jest melodyjka odgrywana podczas inicjalizacji programu przez przycisk pasywny.