AKADEMIA NAUK STOSOWANYCH w ELBLĄGU

Systemy Wbudowane i Mikroprocesory 2024/2025

Raport z budowy robota

Adrian Popielarczyk 21295, Zuzanna Orzechowska 21284

Opis robota

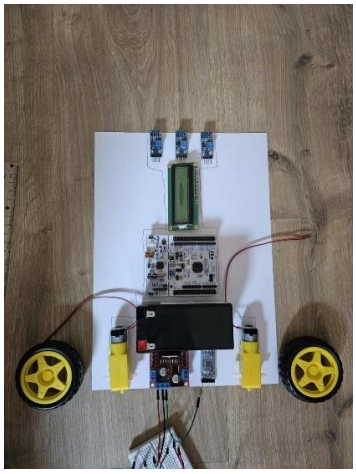
Robot potrafi poruszać się po dowolnym czarnym śladzie na białym tle (linii) z satysfakcjonującą płynnością. Robot posiada ekran LCD, potrafiący wyświetlać parametry oraz napisy. Realizuje te zadanie za pomocą zewnętrznych przycisków. Robot posiada 1 parę osi. Oś nie jest skrętna, a przyczepność oraz napęd występuje na tylnej osi. Skręt 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.

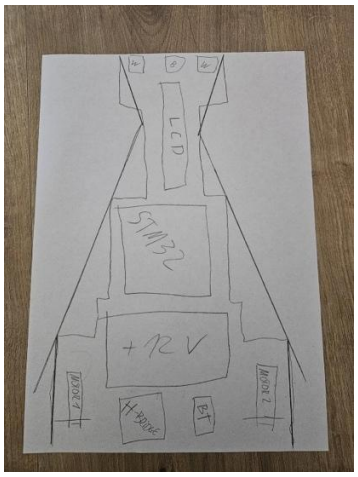
Elementy wybrane do budowy robota

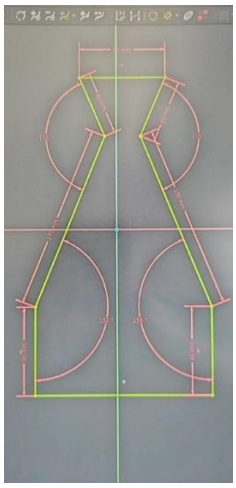
- 1x Płytka STM32 Nucleo L010RB
- 2x Silnik z przekładnią 48:1, o zakresie pracy 3-6V
- 1x Mostek typu H, model L298n
- 1x Akumulator żelowy 12V, 1.3Ah
- 1x Przełącznik On/Off
- Kable do łączeń
- Połowa korka po winie
- Kostka wydrukowana w drukarce 3D

Mechanika robota

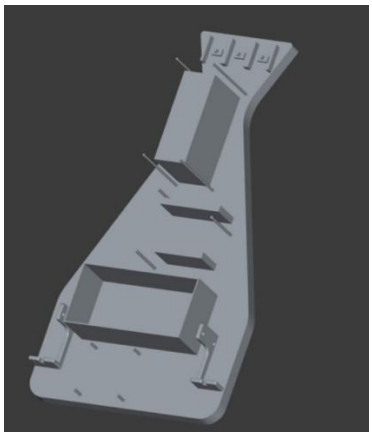
Model koncepcyjny wraz z projektem i budową podwozia, projekt graficzny został wykonany w programie Freecad:



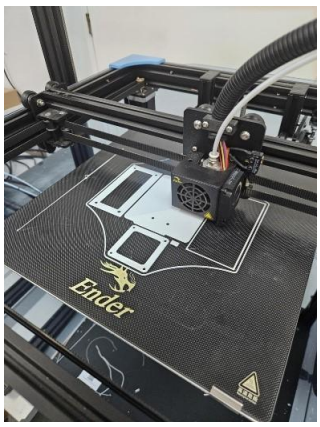


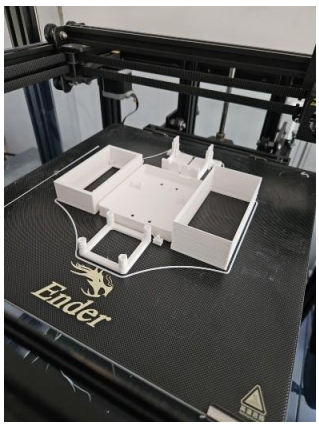


Adrian Popielarczyk 21295, Zuzanna Orzechowska 21284



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

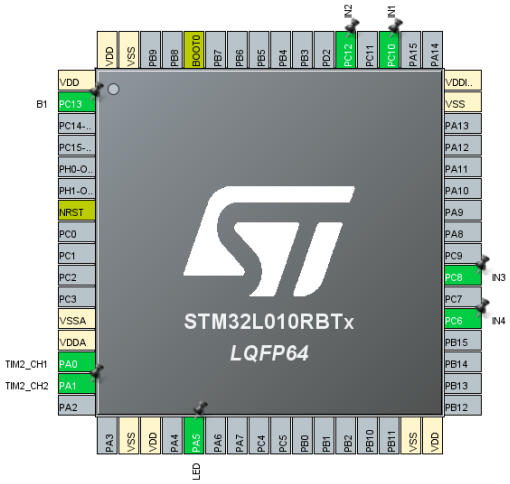
// Przedstawić sposób opracowania mechaniki robota.

Adrian Popielarczyk 21295, Zuzanna Orzechowska 21284

Milestone I – 13.05.2025

Oprogramowanie sterujące

Konfiguracja pinów:



Wyjścia pinów podłączono do odpowiadających deskrypcją pinów na mostku H.

Płytka 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 kolei:

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 rozpoczynanie 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 wyłą-
czone
//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);

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

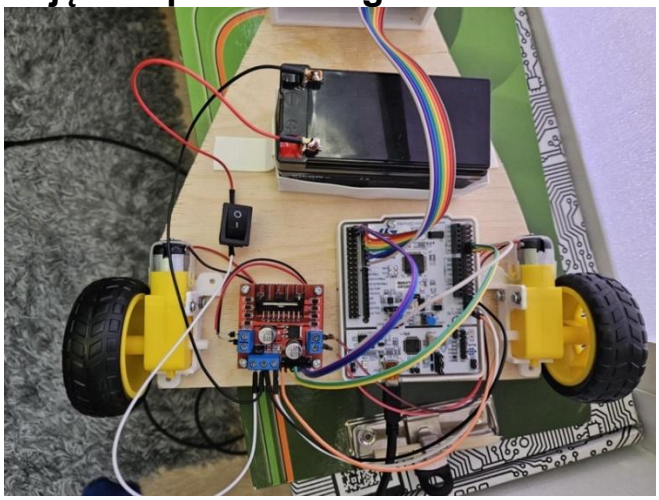
```
HAL_TIM_PWM_Start(&htim2, TIM_CHANNEL_2);

// po inicjalizacji początkowe wartości dla obu kanałów ustawione zostały na 0,
// timer i PWN 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 kofiguracja wskazuje ze sygnał PWM "idzie" w jednym kierunku, co sprawia, ze 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 podłączonego 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, ze sygnał PWM "idzie" w drugim kierunku ka-
nału - 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 motoru, 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); //lewy motor nie dostaje sygnału PWM który sprawiłby, ze ten zacznie się
obracac
__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
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 jed-
nym kierunku
HAL_GPIO_WritePin(GPIOC, IN2_Pin, GPIO_PIN_RESET);
HAL_GPIO_WritePin(GPIOC, IN3_Pin, GPIO_PIN_RESET); // prawy silnik obraca się do tylu, 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 traw 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 zwiększona do ok 95% maksymalnego wypełnie-
nia 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 zatrzymania 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);
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 zgaszą 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 płytki 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 */
```

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 odpowiedzialny za lewy motor
  HAL_GPIO_WritePin(GPIOC, IN2_Pin, GPIO_PIN_RESET); // ustawienie IN2 na niski stan, IN2 to również pin odpowiedzialny za lewy silnik
  HAL_GPIO_WritePin(GPIOC, IN3_Pin, GPIO_PIN_SET); // ustawienie IN3 na wysoki stan, IN3 to pin odpowiedzialny za prawy motor
  HAL_GPIO_WritePin(GPIOC, IN4_Pin, GPIO_PIN_RESET); // ustawienie IN4 na niski stan, IN4 to również 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 pelna moc
  __HAL_TIM_SET_COMPARE(&htim2, TIM_CHANNEL_2, duty); // ustawienie wartości wypełnienia PWM dla kanału 2, czyli prawego silnika na pelna 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 odpowiedzialny za lewy motor
 HAL_GPIO_WritePin(GPIOC, IN2_Pin, GPIO_PIN_RESET); // ustawienie IN2 na niski stan, IN2 to również pin odpowiedzialny za lewy silnik
 HAL_GPIO_WritePin(GPIOC, IN3_Pin, GPIO_PIN_SET); // ustawienie IN3 na wysoki stan, IN3 to pin odpowiedzialny za prawy motor
 HAL_GPIO_WritePin(GPIOC, IN4_Pin, GPIO_PIN_RESET); // ustawienie IN4 na niski stan, IN4 to również 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 pełną moc
 __HAL_TIM_SET_COMPARE(&htim2, TIM_CHANNEL_2, 0); // ustawienie wartości wypełnienia PWM dla kanału 2, czyli prawego silnika na 0, czyli brak ruchu, dzieki temu motor będzie się obracał w lewą stronę
}

void skretWPrawo(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, dzieki 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 noc, 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
}
```

Adrian Popielarczyk 21295, Zuzanna Orzechowska 21284

```
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
}

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 silnika 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 silnika na pełną moc
    __HAL_TIM_SET_COMPARE(&htim2, TIM_CHANNEL_2, duty); // wartość wypełnienia PWM dla prawego silnika na pełną moc
  }
}

/* USER CODE END PFP */
```

Raport z budowy robota

Adrian Popielarczyk 21295, Zuzanna Orzechowska 21284

Kod w sekcji USER CODE 2 – w tej części zaczęto prace nad działaniem wyświetlacza LCD, działa on „po części”, to znaczy łączy się z płytką 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 wyświetlacz LCD
 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
      }
```

```
    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 skręci
        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 skręcić
        skretWPrawo(); // 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 skręcić
        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 skręcić
        skretWPrawo(); // 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 */
}
```

/* USER CODE BEGIN WHILE */

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

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 zainicjo-
wać zapis danych
  HAL_Delay(1); // zwiększony czas
  HAL_GPIO_WritePin(LCD_EN_GPIO_Port, LCD_EN_Pin, GPIO_PIN_RESET); // Ustawienie linii Enable na stan niski, aby zakoń-
czyć zapis danych
```

Adrian Popielarczyk 21295, Zuzanna Orzechowska 21284

```
    HAL_Delay(1); // zwiększony 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)
    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++); // Wysłanie aktualnego znaku do wyświetlacza LCD i przejście do następnego znaku
    }
}

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
}
```

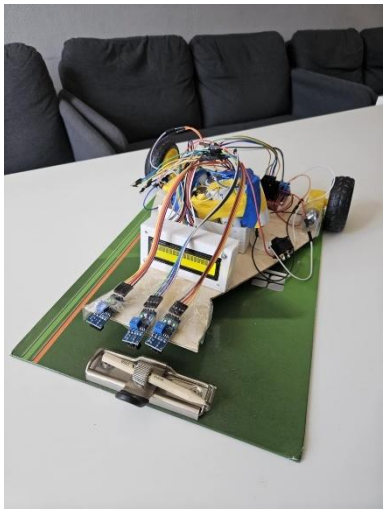
Poniżej znajduje się opis kodu w własnej bibliotece do wyświetlacza – plik lcd_hd44780.h, zawiera on głównie deklaracje odpowiednio podłączonych pinów oraz inicjalizację wykorzystanych funkcji.:

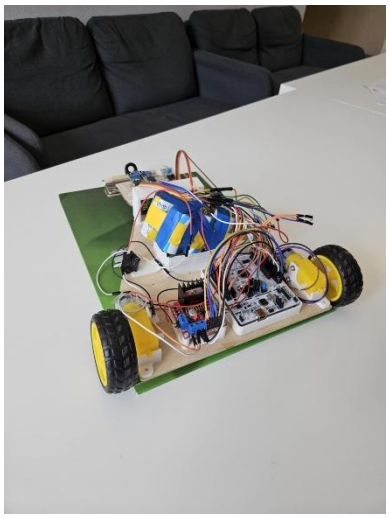
```
#ifndef __LCD_HD44780_H__ // dyrektywa preprocesora, aby uniknąć wielokrotnego dołączenia pliku nagłówkowego
#define __LCD_HD44780_H__ // definicja makra __LCD_HD44780_H__, które jest używane do ochrony przed wielokrotnym dołączeniem tego pliku nagłówkowego

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

// 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
```

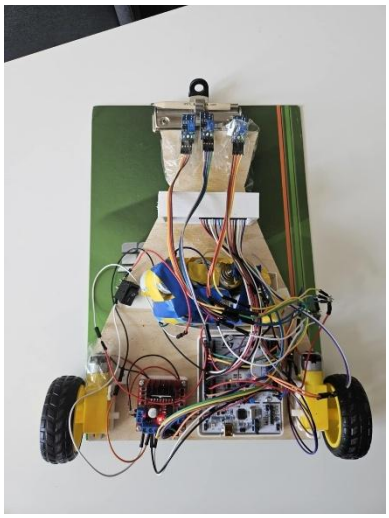
Adrian Popielarczyk 21295, Zuzanna Orzechowska 21284

```
#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:







Wnioski:

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