Projekt

Sterowniki robotów

Dokumentacja

Zdalnie sterowany mobilny robot holonomiczny *Waddles*

Skład grupy:
Emilia Szymańska, 248975

Termin: ponTP17

Prowadzący:
dr inż. Wojciech Domski

7 czerwca 2021

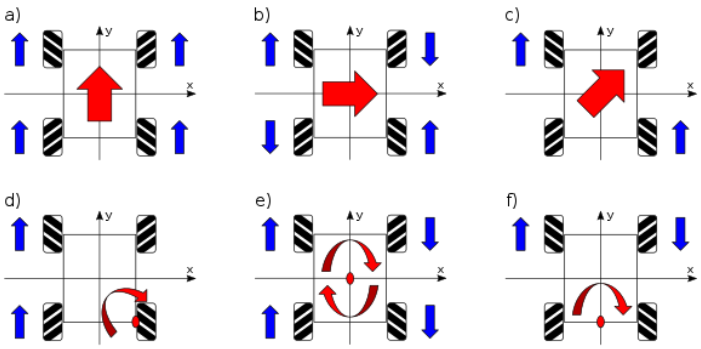
Spis treści

1 Opis projektu

W szerszej perspektywie, projekt ma na celu zaprojektowanie, zbudowanie i zaprogramowanie holonomicznego robota wykorzystującego koła mecanum (rys. 1). Robot ten będzie zdalnie sterowany za pomocą aplikacji zainstalowanej na urządzeniu z systemem Android. Na projekcie ze Sterowników Robotów główny nacisk zostanie postawiony na stworzenie działającego systemu sterowania silnikami prądu stałego z enkoderami z wykorzystaniem standardu Bluetooth do komunikacji. Dodatkowo, czujnik temperatury (ze standardem I2C/TWI) umieszczony w pobliżu układu elektronicznego zastosowany zostanie do monitorowania temperatury i zatrzymania pracy systemu w przypadku przegrzewania się jego elementów. Czujnik odległości z kolei, po wykryciu przeszkody z przodu robota, zablokuje możliwość wykonania komendy jazdy do przodu (jeśli takowa zostanie wydana przez użytkownika).

Rysunek 1: Przykładowy robot wykorzystujący koła mecanum; źródło: [1]

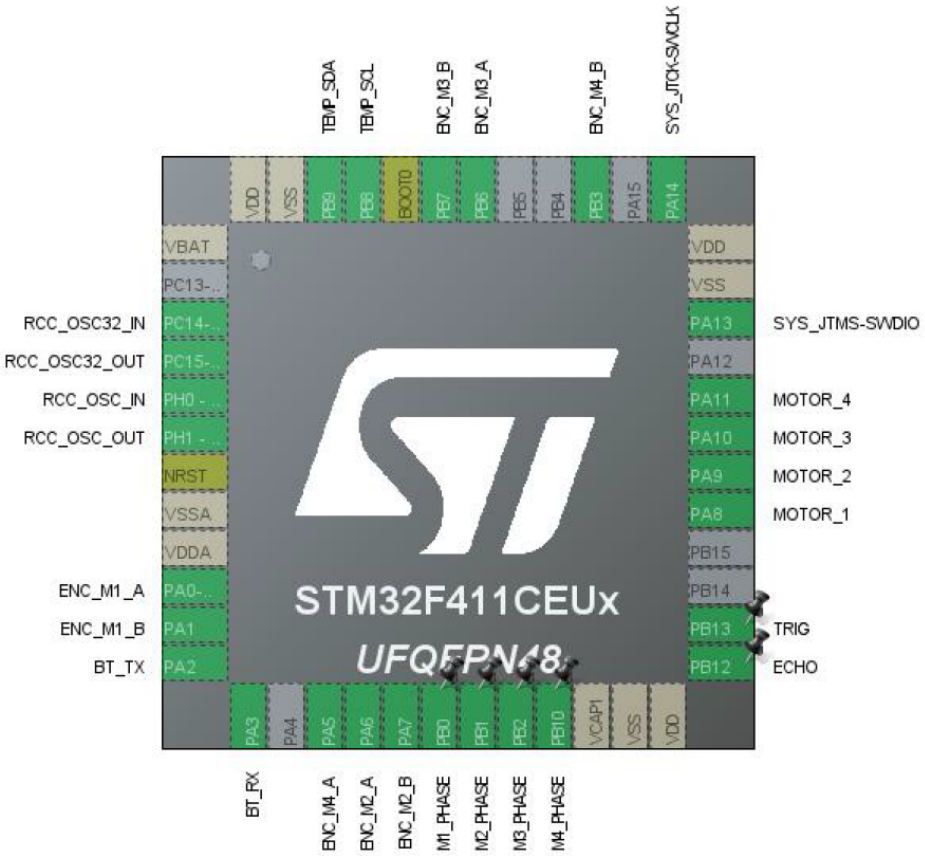
Zastosowanie kół mecanum w robocie mobilnym sprawia, że liczba stopni swobody dostępna dla robota jest równa liczbie stopni swobody przez niego kontrolowanych. Sterowanie takim robotem (opisane szczegółowiej m.in. w [2]) wymaga odpowiedniego skoordynowania kierunku i prędkości obrotu wszystkich czterech kół zgodnie z zasadą przedstawioną na rys. 2. Mobilne urządzenie stanowi lekką i poręczną alternatywę dla kontrolera, dlatego na wzór [3] planowane jest wykorzystanie komunikacji po Bluetooth z aplikacją na systemie Android.



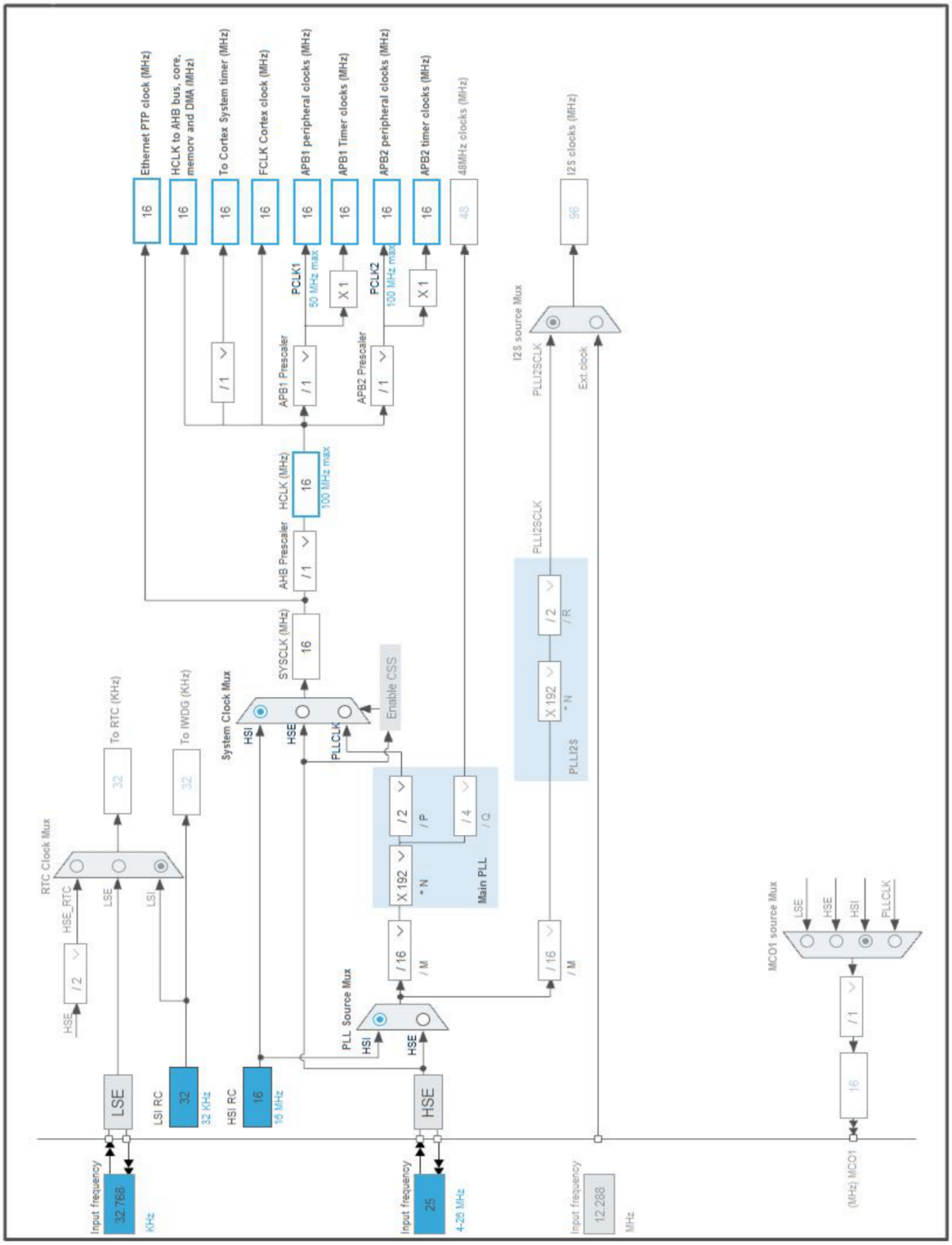
Rysunek 2: Sterowanie robotem z kołami mecanum; źródło: [1]

2 Konfiguracja mikrokontrolera

Do projektu zostanie wykorzystany mikrokontroler STM32F411CEU (Blackpill) – nota katalogowa [4].



Rysunek 3: Konfiguracja wyjść mikrokontrolera w programie STM32CubeMX



Rysunek 4: Konfiguracja zegarów mikrokontrolera

2.1 Konfiguracja pinów

Numer pinu	PIN	Tryb pracy	Funkcja/etykieta
3	PC14-OSC32_IN	RCC_OSC32_IN	
4	PC15-OSC32_OUT	RCC_OSC32_OUT	
5	PH0 - OSC_IN	RCC_OSC_IN	
6	PH1 - OSC_OUT	RCC_OSC_OUT	
10	PA0-WKUP	TIM5_CH1	ENC_M1_A
11	PA1	TIM5_CH2	ENC_M1_B
12	PA2	USART2_TX	BT_TX
13	PA3	USART2_RX	BT_RX
15	PA5	TIM2_CH1	ENC_M4_A
16	PA6	TIM3_CH1	ENC_M2_A
17	PA7	TIM3_CH2	ENC_M2_B
18	PB0	GPIO_Output	M1_PHASE
19	PB1	GPIO_Output	M2_PHASE
20	PB2	GPIO_Output	M3_PHASE
21	PB10	GPIO_Output	M4_PHASE
25	PB12	GPIO_EXTI12	ECHO
26	PB13	GPIO_Output	TRIG
29	PA8	TIM1_CH1	MOTOR_1
30	PA9	TIM1_CH2	MOTOR_2
31	PA10	TIM1_CH3	MOTOR_3
32	PA11	TIM1_CH4	MOTOR_4
34	PA13	SYS_JTMS-SWDIO	
37	PA14	SYS_JTCK-SWCLK	
39	PB3	TIM2_CH2	ENC_M4_B
42	PB6	TIM4_CH1	ENC_M3_A
43	PB7	TIM4_CH2	ENC_M3_B
45	PB8	I2C1_SCL	TEMP_SCL
46	PB9	I2C1_SDA	TEMP_SDA

Tabela 1: Konfiguracja pinów mikrokontrolera

2.2 USART

Interfejs szeregowy wykorzystany jest do komunikacji za pomocą Bluetooth. Konfiguracja jest pokazana w tabeli 2.

Parametr	Wartość
Baud Rate	9600
Word Length	8 Bits (including parity)
Parity	None
Stop Bits	1

Tabela 2: Konfiguracja peryferium USART

2.3 I2C

Komunikacja według protokołu I2C (skonfigurowanego jak w tab. 3) służy do porozumiewania się z czujnikiem temperatury.

Parametr	Wartość
I2C Speed Mode	Standard Mode
Word Length	8 Bits (including parity)
I2C Clock Speed [Hz]	100 000
Clock No Stretch Mode	Disabled
Primary Address Length selection	7-bit
Dual Address Acknowledged	Disabled
Primary slave address	0
General Call address detection	Disabled

Tabela 3: Konfiguracja peryferium I2C

2.4 PWM

Pulse Width Modulation - skonfigurowane na 4 kanałach licznika TIM1 jak w 4 - służy do sterowania czteroma silnikami.

Parametr	Wartość
Prescaler (PSC - 16 bits value)	159
Counter Mode	UP
Counter Period (AutoReload Register - 16 bits value)	99
Internal Clock Division (CKD)	No Division
Repetition Counter (RCR - 8 bits value)	0
auto-reload preload	Disabled

Tabela 4: Konfiguracja PWM na TIM1

2.5 Encoder mode

Cztery liczniki zostały przeznaczone na otrzymywanie danych z enkoderów. TIM2 oraz TIM5 są 32-bitowe tab. (5), natomiast TIM3 i TIM4 są 16-bitowe (tab. 6). Ilość tików zliczona na enkoderach jest odczytywana z częstotliwością 100 Hz (według TIM9).

Parametr	Wartość
Prescaler (PSC - 16 bits value)	0
Counter Mode	UP
Counter Period (AutoReload Register - 32 bits value)	4294967295
Internal Clock Division (CKD)	No Division
auto-reload preload	Disabled
Encoder Mode	Encoder Mode TI1 and TI2
Polarity	Rising Edge
IC Selection	Direct
Prescaler Division Ratio	No division
Input Filter	15

Tabela 5: Konfiguracja TIM2 oraz TIM5 dla enkoderów.

Parametr	Wartość
Prescaler (PSC - 16 bits value)	0
Counter Mode	UP
Counter Period (AutoReload Register - 16 bits value)	65535
Internal Clock Division (CKD)	No Division
auto-reload preload	Disabled
Encoder Mode	Encoder Mode TI1 and TI2
Polarity	Rising Edge
IC Selection	Direct
Prescaler Division Ratio	No division
Input Filter	15

Tabela 6: Konfiguracja TIM3 oraz TIM4 dla enkoderów.

2.6 Pozostałe liczniki - TIM9, TIM10 oraz TIM11

TIM9 (tab. 7) służy jako podstawa czasu dla enkoderów - z częstotliwością 100 Hz wywoływane jest przerwanie od przepełnienia licznika, a tym samym obliczana jest ilość obrotów wałów silników na minutę.

Parametr	Wartość
Mode	clock source
Prescaler (PSC - 16 bits value)	99
Counter Mode	UP
Counter Period (AutoReload Register - 16 bits value)	1599
Internal Clock Division (CKD)	No Division
auto-reload preload	Disabled

Tabela 7: Konfiguracja TIM9

Licznik TIM10 (tab. 8) służy do zmiany wartości flagi, od której zależy odczyt temperatury oraz odległości do najbliższej przeszkody. Dzięki temu nie ma potrzeby ciągłego wywoływania tych funkcji ani użycia HAL_Delay(). Częstotliwość ustawiona została na 1 Hz.

Parametr	Wartość
Mode	activated
Prescaler (PSC - 16 bits value)	15 999
Counter Mode	UP
Counter Period (AutoReload Register - 16 bits value)	999
Internal Clock Division (CKD)	No Division
auto-reload preload	Disabled

Tabela 8: Konfiguracja TIM10

Celem wykorzystania licznika TIM11 jest pomiar długości trwania sygnału odbieranego na pinie ECHO z ultradźwiękowego czujnika odległości.

Parametr	Wartość
Mode	activated
Prescaler (PSC - 16 bits value)	15
Counter Mode	UP
Counter Period (AutoReload Register - 16 bits value)	65535
Internal Clock Division (CKD)	No Division
auto-reload preload	Disabled

Tabela 9: Konfiguracja TIM11

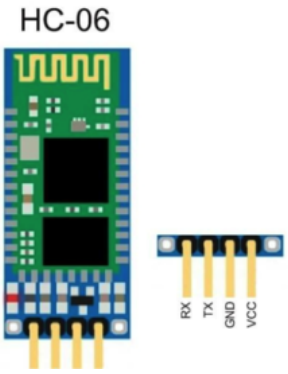
3 Urządzenia zewnętrzne

3.1 Moduł Bluetooth

Użyty moduł (HC-06) [5] umożliwia komunikację w ramach portu szeregowego z innym urządzeniem. Docelowo tym urządzeniem będzie telefon z oprogramowaniem Android, jednak na czas rozwoju testowana jest funkcjonalność przy pomocy laptopa z programem RealTerm. Na początku zdecydowałam się zmienić nazwę urządzenia, więc wykorzystując konwerter USB-UART połączyłam się z modułem i wpisując w terminalu komendę *AT+NAMEWaddles* udało się spersonalizować nazwę (Rys.5). Nie można wywoływać komend AT po połączeniu się z urządzeniem przy pomocy Bluetootha, dlatego konwerter był konieczny, by móc cokolwiek zmienić.

Rysunek 5: Widoczne urządzenie z perspektywy komputera

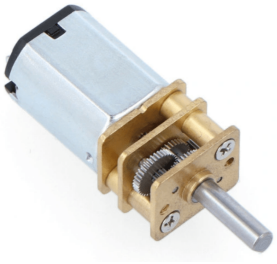
Prezentacja modułu znajduje się na Rys.6. Wymaga on połączenia pinów RX do BT_TX na STM32 oraz TX do BT_RX na STM32. Podane zasilanie było na poziomie 3V3. Konfiguracja protokołu USART znajduje się w sekcji 2.2.



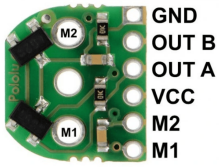
Rysunek 6: Moduł Bluetooth HC06. Źródło: [6].

3.2 Sterowniki silników, enkodery i silniki

Mikro silniki prądu stałego firmy Pololu z przekładnią 150:1 oraz prędkością obrotową 200 obr/min (Rys.7) zostały połączone z enkoderami magnetycznymi (Rys.8) tego samego producenta. Silniki zasilone zostały napięciem 8V, logika enkoderów natomiast napięciem 3V3.

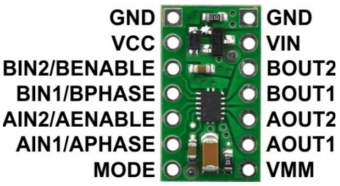


Rysunek 7: Silniki Pololu HPCB. Źródło: [7].



Rysunek 8: Enkodery magnetyczne Pololu. Źródło: [8].

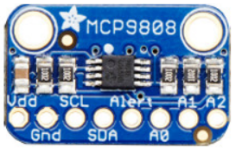
Nad sterowaniem silnikami czuwa dwukanałowy sterownik silników DRV8835 ([9], Rys. 9). Sterując sygnałem PWM (piny ENABLE połączone z MOTOR_x) oraz podając odpowiednie wyjście na GPIO (piny PHASE połączone z Mx_PHASE) można sterować prędkością obrotową wału oraz kierunkiem obrotu dwóch silników. Należy połączyć piny mikrokontrolera ENC_Mx_A, ENC_Mx_B do pinów enkodera OUT A oraz OUT B. W przypadku połączenia silników do sterownika, piny xOUT1, xOUT2 muszą być połączone z pinami enkodera M1 oraz M2. Pin MODE decyduje o trybie pracy sterownika - zwarłam go z napięciem 3V3.



Rysunek 9: Dwukanałowy sterownik silników DRV8835 Pololu. Źródło: [10].

3.3 Czujnik temperatury

Czujnik temperatury MCP9808 firmy Adafruit (Rys. 10, [11]) potrafi zmierzyć temperaturę od -40^{o}C do +125^{o}C z dokładnością do +0.0625^{o}C. Dane pomiarowe wysyłane są przy pomocy protokołu I2C, gdzie domyślnym adresem urządzenia jest 0x18. Można adres zmienić odpowiednio przy użyciu pinów A0, A1 i A2, jednak było to tutaj zbędne. Czujnik operuje zarówno na logice 3V3, jak i 5V.

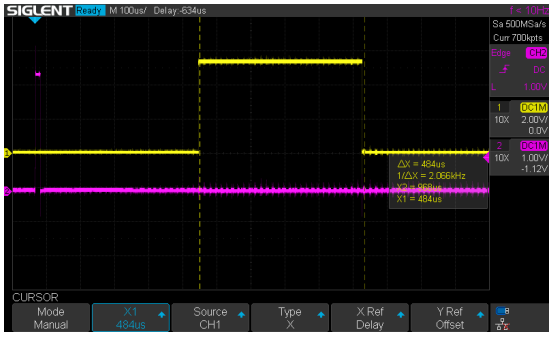


Rysunek 10: Czujnik temperatury wysokiej precyzji MCP9808. Źródło: [12].

3.4 Czujnik odległości

Użyty czujnik odległości HCSR-04 ([13], Rys. 11) wykorzystuje fale ultradźwiękowe do określenia dystansu do najbliższej przeszkody. Wysłanie fali następuje wskutek podania na pinie TRIG impulsu HIGH o długości 10 μs, a czas trwania sygnału prostokątnego otrzymanego na pinie ECHO jest wprost proporcjonalny do odległości (przykład na rys. 12a, gdzie kanał pierwszy jest sygnałem ECHO, a kanał drugi sygnałem TRIG). Czas ten należy przemnożyć przez prędkość rozchodzenia się dźwięku i podzielić przez 2, by otrzymać wartość dystansu. Czujnik zasilany jest z napięcia 5V, więc konieczne było zastosowanie przetwornicy STEP-DOWN, by z napięcia na silnikach otrzymać żądaną wartość. Zakres pracy czujnika wynosi 2cm-400cm, choć przy krańcowych wartościach pojawiają się znaczne zakłócenia.

Rysunek 11: Ultradźwiękowy czujnik odległości HCSR-04. Źródło: [14].



(a) Inicjowanie odczytu.

(b) Zakłócenia sygnału.

Rysunek 12: Odczyty sygnałów na oscyloskopie związanych z czujnikiem odległości.

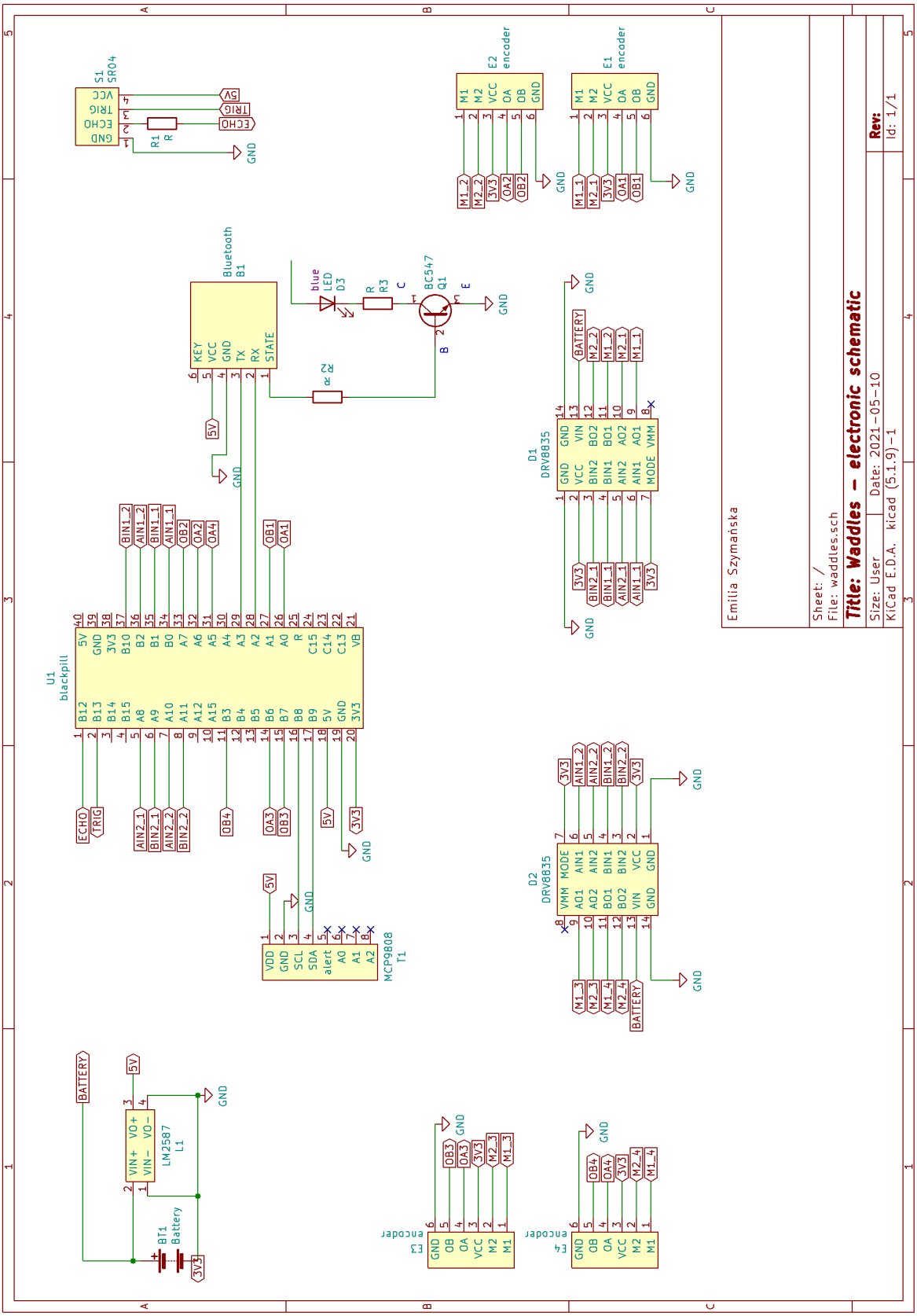
Czujnik ultradźwiękowy potrafi jednak zwracać co jakiś czas losowe wartości, co widać na rysunku 12b - mimo braku zmiany odległości część sygnałów ECHO jest krótsza, a część dłuższa. W związku z tym do analizy odległości wykorzystuję średnią z ostatnich ośmiu próbek:

```
uint8_t measurements = 8;
uint16_t distance_array[8];
uint16_t distance = 0;
uint16_t average_distance = 0;
uint8_t current_index = 0;
uint8_t distance_sum = 0;

distance = USONIC_get_distance();
for(uint8_t i = 0; i < measurements; i++)
    distance_array[i] = distance;
current_index = 0;
distance_sum = measurements * distance;
average_distance = distance;
.....
while (1)
  {
    if(main_flag)
    {
      .....
      distance = USONIC_get_distance();
      distance_sum = distance_sum - distance_array[current_index] +  distance;
      distance_array[current_index] = distance;
      current_index = (current_index + 1) % measurements;
      average_distance = distance_sum / measurements;
      .....
    }
    .....
  }
```

4 Projekt elektroniki

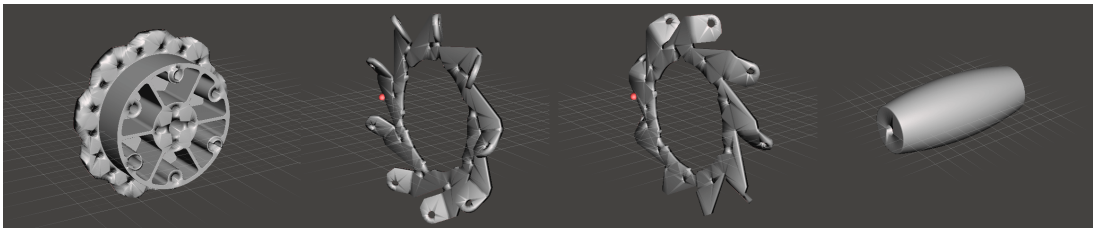
Schemat elektroniki i połączeń przedstawiony został na rys. 13.



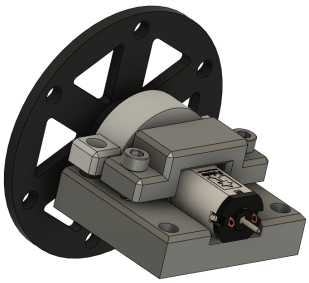
Rysunek 13: Schemat elektroniczny robota.

5 Konstrukcja mechaniczna

Modele kół oraz elementów przytrzymujących silniki zostały zaczerpnięte z portalu Thingiverse - zdjęcia modeli zostały przedstawione odpowiednio na rys. 14 i 15.

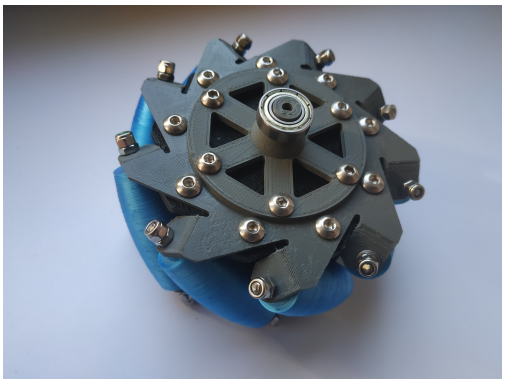


Rysunek 14: Modele kół mecanum.

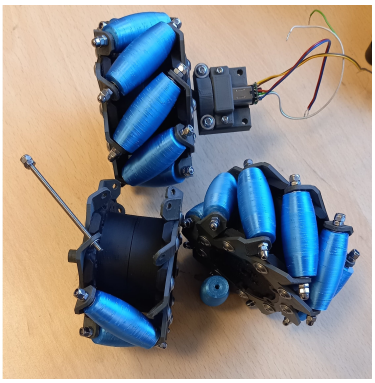


Rysunek 15: Modele elementów przytrzymujących silniki.

Model wymagał wydrukowania na drukarce 3D, powiercenia otworów oraz rolek i gwintowania niektórych otworów. Do złożenia kół potrzebne były łożyska, śruby (M3 i M4) z łbem walcowym, nakrętki, podkładki (w tym też sprężyste) i pręt gwintowany (później pocięty na mniejsze części). Proces składania kół widoczny jest na rys. **??**.



(a) Złożone koło.

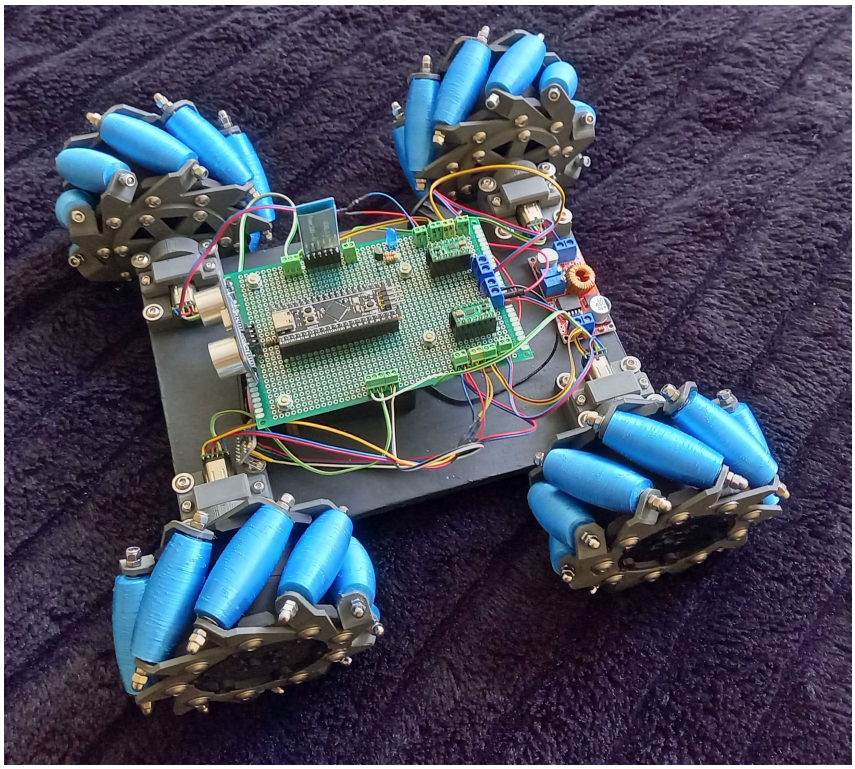


(b) Montaż kół.

Rysunek 16: Gotowe koła mecanum.

Podstawa robota jest wycięta ze sklejki i pomalowana. Wywiercone zostały otwory umożliwiające montaż silników i dystanserów mocujących płytkę uniwersalną z układem elektronicznym.

Po zamocowaniu wszystkich elementów, robot wygląda jak na rys. 17.



Rysunek 17: Złożony robot.

6 Opis działania programu

Program (dostępny pod repozytorium `https://github.com/emilia-szymanska/waddles_stm32`) podzielony został na moduły z odpowiednimi funkcjami.

1. Moduł bluetooth.h/.c:
   - void **BT_start()** - załącza przerwania na porcie szeregowym;
   - void **BT_stop()** - wyłącza przerwania na porcie szeregowym;
   - void **BT_send_message(uint8_t *data)** - wysyła dane *data* za pośrednistwem portu szeregowego;
   - void **HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart)** - definiuje ciało przerwania po otrzymaniu danych na porcie szeregowym (zapisuje te dane w odpowiedniej zmiennej globalnej i ustawia odpowiednią flagę na 1).
2. Moduł pid.h/.c:
   - void **PID_init(PID_handler *pid, float p, float i, float d)** - inicjuje regulator PID względem nastaw podanych jako parametry;
   - uint32_t **PID_output(PID_handler *pid, uint32_t desired_value, uint32_t measured_value)** - wylicza wyjście z regulatora PID na podstawie danej instancji PIDa, wartości zadanej i wartości zmierzonej (z enkodera).
3. Moduł encoders.h/.c:
   - void **ENCODERS_start()** - załącza liczniki działające w trybie Encoder mode oraz przerwanie od licznika, który wywołuje odczytanie danych z enkoderów;
   - void **ENCODERS_stop()** - wyłącza liczniki działające w trybie Encoder mode oraz przerwania od licznika, który wywołuje odczytanie danych z enkoderów;
   - void **ENCODERS_get_rpm()** - odpowiednio przetwarza dane z enkoderów, by w zmiennych globalnych zapisać wartości prędkości obrotowych (w obrotach na minutę).

4. Moduł motors.h/.c:

   - void **MOTORS_start()** - ustawia i załącza sygnały PWM, ustawia fazy na stan wysoki;
   - void **MOTORS_stop()** - wyłącza sygnały PWM;
   - void **MOTORS_set_speed(uint8_t motor_nr, uint8_t speed)** - dla danego silnika (numerowane od 1 do 4) ustawia wypełnienie PWM według wartości procentowej podanej jako zmienna *speed*;
   - void **MOTORS_set_polarity(uint8_t motor_nr, uint8_t polarity)** - ustawia polaryzację (fazę) wybranego silnika (numerowanego od 1 do 4) na stan HIGH lub LOW.
   - void **MOTORS_go_forward()** - pojazd porusza się przed siebie;
   - void **MOTORS_go_backward()** - pojazd porusza się do tyłu;
   - void **MOTORS_go_left()** - pojazd porusza się w lewo;
   - void **MOTORS_go_right()** - pojazd porusza się w prawo;
   - void **MOTORS_rotate()** - pojazd obraca się w miejscu zgodnie z ruchem wskazówek zegara;
   - void **MOTORS_go_forwardleft()** - pojazd porusza się po przekątnej (w lewo przed siebie);
   - void **MOTORS_go_forwardright(** - pojazd porusza się po przekątnej (w prawo przed siebie);
   - void **MOTORS_go_backwardright()** - pojazd porusza się po przekątnej (w prawo w tył);
   - void **MOTORS_go_backwardleft()** - pojazd porusza się po przekątnej (w lewo w tył);
   - void **MOTORS_M1_forward()** - silnik nr 1 kręci się do przodu;
   - void **MOTORS_M2_forward()** - silnik nr 2 kręci się do przodu;
   - void **MOTORS_M3_forward()** - silnik nr 3 kręci się do przodu;
   - void **MOTORS_M4_forward()** - silnik nr 4 kręci się do przodu;
   - void **MOTORS_M1_backward()** - silnik nr 1 kręci się do tyłu;
   - void **MOTORS_M2_backward()** - silnik nr 2 kręci się do tyłu;
   - void **MOTORS_M3_backward()** - silnik nr 3 kręci się do tyłu;
   - void **MOTORS_M4_backward()** - silnik nr 4 kręci się do tyłu;
   - void **setpoints_to_zeros(uint16_t *setpoints_array)** - ustawia wszystkie wartości zadane w tablicy na 0;
   - void **setpoints_to_const(uint16_t *setpoints_array)** - ustawia wszystkie wartości zadane w tablicy na ustaloną wartość stałą;
   - void **setpoints_to_fr_bl(uint16_t *setpoints_array)** - ustawia wartości zadane w taki sposób, by robot mógł poruszać się po przekątnej (do przodu w prawo lub do tyłu w lewo);
   - void **setpoints_to_fl_br(uint16_t *setpoints_array)** - ustawia wartości zadane w taki sposób, by robot mógł poruszać się po przekątnej (do przodu w lewo lub do tyłu w prawo).

5. Moduł ultrasonic_sensor.h/.c:

   - void **USONIC_start()** - ustawia czytanie pinu ECHO na rosnące zbocze i załączenie licznika tim11;
   - void **USONIC_stop()** - zatrzymuje licznik tim11;
   - uint16_t **USONIC_get_distance** - wysyła sygnał prostokątny o długości 10 mikrosekund i odczytuje wartości na pinie ECHO;
   - void **delay(uint16_t time)** - funkcja opóźniająca, której argument podawany jest w mikrosekundach;
   - void **set_to_rising_edge()** - ustawia przerwania na zbocze rosnące na pinie ECHO;
   - void **set_to_falling_edge()** - ustawia przerwania na zbocze opadające na pinie ECHO;
   - void **HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)** - definiuje reakcję na przerwania od zmiany zbocza na pinie ECHO (oblicza długość trwania sygnału).

6. Moduł temperature_sensor.h/.c:

   - float **TEMP_get_data()** - zwraca zmiennoprzecinkową wartość temperatury odczytaną po wysłaniu zapytania do czujnika według protokołu I2C.

7. Funkcje dotyczące czujnika odległości w funkcji main.c (dopiero działające w stu procentach kawałki kodu przenoszone są do oddzielnych plików):

   - void **delay(uint16_t time)** - pozwala na opóźnienie programu o daną liczbę mikrosekund;
   - void **HCSR04_Read (void)** - wysyła załączający sygnał na pin TRIG i następnie czeka na odczytanie wartości odległości;
   - void **HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)** - przerwanie to umożliwia zareagowanie na zbocze występujące na pinie ECHO, a tym samym obliczenie długości trwania sygnału.

6.1 Główna pętla programu

W głównej pętli programu można wyróżnić trzy sekcje, które są wykonywane w zależności od tego, czy odpowiadająca im flaga jest podniesiona.

6.1.1 Odpowiednie reagowanie na komendy wysyłane przez aplikację

Po odebraniu danych przez moduł Bluetooth podnoszona jest flaga *command_flag*, która pozwala na interpretację komend. Przy jeździe wprzód sprawdzany jest warunek, czy odległość do najbliższej przeszkody jest odpowiednio duża - w przeciwnym wypadku pojazd się zatrzyma i nie pozwoli na dalszą jazdę wprzód.

```
while (1)
  {
    .....
    if(command_flag)
    {
      if(strcmp(command, "nn") == 0)
      {
        setpoints_to_zeros(setpoints);
      }
      else if(strcmp(command, "ll") == 0)
      {
          setpoints_to_const(setpoints);
          MOTORS_go_left();
      }
      else if(strcmp(command, "rr") == 0)
      {
          setpoints_to_const(setpoints);
          MOTORS_go_right();
      }
      else if(strcmp(command, "bb") == 0)
      {
          setpoints_to_const(setpoints);
          MOTORS_go_backward();
      }
      else if(strcmp(command, "ff") == 0)
      {
        if(average_distance <= DISTANCE_LIMIT)
        {
            setpoints_to_zeros(setpoints);
        }
        else
        {
          setpoints_to_const(setpoints);
          MOTORS_go_forward();
        }
      }
```

```
37        else if(strcmp(command, "fl") == 0)
38        {
39          if(average_distance <= DISTANCE_LIMIT)
40          {
41              setpoints_to_zeros(setpoints);
42          }
43          else
44          {
45            setpoints_to_fl_br(setpoints);
46            MOTORS_go_forwardleft();
47          }
48        }
49        else if(strcmp(command, "fr") == 0)
50        {
51          if(average_distance <= DISTANCE_LIMIT)
52          {
53            setpoints_to_zeros(setpoints);
54          }
55          else
56          {
57            setpoints_to_fr_bl(setpoints);
58                MOTORS_go_forwardright();
59          }
60        }
61        else if(strcmp(command, "bl") == 0)
62        {
63            setpoints_to_fr_bl(setpoints);
64            MOTORS_go_backwardleft();
65        }
66        else if(strcmp(command, "br") == 0)
67        {
68            setpoints_to_fl_br(setpoints);
69            MOTORS_go_backwardright();
70        }
71        else if(strcmp(command, "cc") == 0)
72        {
73            MOTORS_rotate();
74            setpoints_to_const(setpoints);
75        }
76        command_flag = 0;
77      }
78      .....
79    }
```

6.1.2 Regulacja prędkości kół

Flaga *control_flag* odpowiada za cykliczne (co 10 ms liczone przez TIM9) odczytywanie wartości z enkoderów i odpowiednie podawanie wartości na koła po przepuszczeniu danych przez regulator PID.

```
1  while (1)
2    {
3      .....
4      if(control_flag)
5      {
6        ENCODERS_get_rpm();
7        MOTORS_set_speed(1, PID_output(&pid_m1, setpoints[0], E1_rpm));
8        MOTORS_set_speed(2, PID_output(&pid_m2, setpoints[1], E2_rpm));
9        MOTORS_set_speed(3, PID_output(&pid_m3, setpoints[2], E3_rpm));
10       MOTORS_set_speed(4, PID_output(&pid_m4, setpoints[3], E4_rpm));
11       control_flag = 0;
12     }
13     .....
14   }
```

6.1.3 Odczyt temperatury i odległości

Pierwsza sekcja (zależna od flagi *main_flag* wystawianej od przerwania licznika TIM10 z częstotliwością 1 Hz) odpowiada za czytanie temperatury i odległości do najbliższej przeszkody z czujników. Jeśli temperatura jest wyższa niż zdefiniowana wartość *TEMPERATURE_ALERT*, wówczas uruchamiana jest procedura wyłączająca wszystkie peryferia i odczytująca w pętli temperaturę. Jeśli temperatura spadnie, peryferia są na nowo uruchamiane i następuje wyjście z procedury. Sposób i powód takiego obliczania odległości, jaki jest zaprezentowany poniżej, wytłumaczony jest w sekcji dotyczącej czujnika ultradźwiękowego.

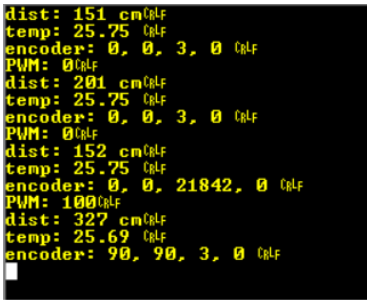
```
while (1)
  {
    .....
    if(main_flag)
    {
      temperature = TEMP_get_data();
      if(temperature >= TEMPERATURE_ALERT)
      {
          setpoints_to_zeros(setpoints);
        temperature_alert();
      }
      distance = USONIC_get_distance();
      distance_sum = distance_sum - distance_array[current_index] +  distance;
      distance_array[current_index] = distance;
      current_index = (current_index + 1) % measurements;
      average_distance = distance_sum / measurements;
      main_flag = 0;
    }
    .....
  }
```

6.2 Funkcje pomocnicze

W pliku main.c znajdują się również cztery funkcje pomocnicze:

- void **HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)** - podnoszenie odpowiednich flag w zależności od przerwania od liczników,
- void **start_peripherals()** - załączenie peryferiów (Bluetooth, czujnik odległości, silniki, enkodery, przerwania),
- void **start_peripherals()** - wyłączenie peryferiów (głównie przerwań, zatrzymanie silników i wyłączenie PWM),
- void **temperature_alert()** - obsługa alarmu od czujnika temperatury (wyłączenie peryferiów i ciągły odczyt temperatury, aż nie spadnie ona poniżej temperatury alarmowej).

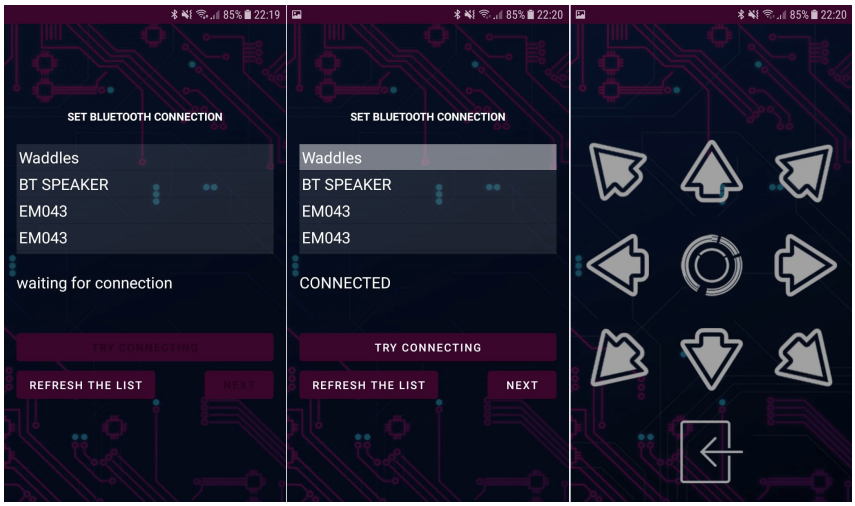
Przykład testowania programu został zaprezentowany na rys. 18. Co ważne, dwa ostatnie silniki nie były połączone, zatem wartości otrzymywane z tych enkoderów są wartości zakłóceń na pinach. Jest to przykład testowania przed dodaniem regulatora PID - wysyłanie tylu znaków przez Bluetooth, gdy chcemy regulować prędkości kół co 10 ms, sprawiało, że robot co chwila przestawał kręcić kołami.



Rysunek 18: Wynik działania programu w konsoli RealTerm.

7 Sterowanie

Sterowanie odbywa się poprzez otrzymywanie dwuliterowych komend z aplikacji. Sama aplikacja została zaimplemetowana z wykorzystaniem języka Java z Androidowym API. Składa się ona z dwóch części - okna wyboru urządzenia i połączenia się z nim oraz okna wyboru komend pod postaciami strzałek. Wygląd aplikacji zaprezentowany jest na rys. 19.



Rysunek 19: Kolejne etapy łączenia się z robotem w aplikacji.

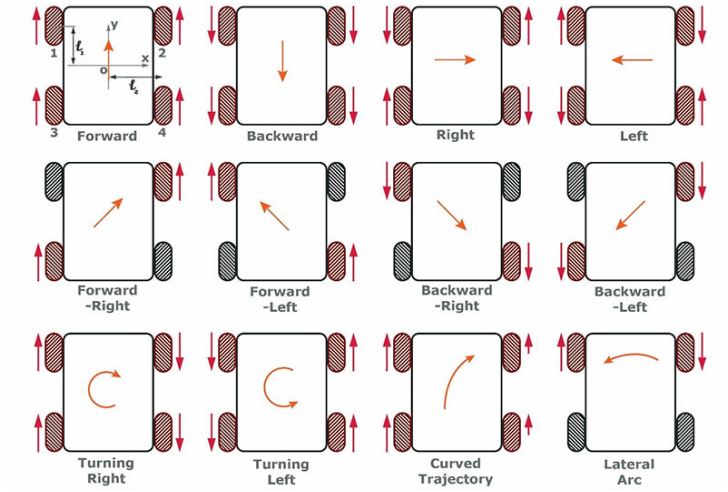
Przy wybraniu odpowiedniej strzałki, w zmiennej zapisywany jest odpowiednia dwuliterowa komenda, która co 20 milisekund wysyłana jest przez połączenie Bluetooth. Przy braku wciśnięcia przycisku jest ustawiana wartość domyślna.

Iterpretacja komend jest następująca:

- **nn** - brak wybranej opcji, zatrzymanie pojazdu;
- **ff** - poruszanie się wprzód,
- **bb** - poruszanie się w tył,
- **ll** - poruszanie się w lewo,
- **rr** - poruszanie się w prawo,
- **fl** - poruszanie się w lewo wprzód,
- **fr** - poruszanie się w prawo wprzód,
- **bl** - poruszanie się w lewo w tył,
- **br** - poruszanie się w prawo w tył,
- **cc** - obracanie się w miejszu zgodnie z ruchem wskazówek zegara.

Sposób sterowania silnikami jest przedstawiony na rys. 20. W przypadku mojego robota koła numerowane są następująco: 1 (prawe tylne), 2 (prawe przednie), 3 (lewe przednie), 4 (prawe tylne). Pod spodem każdej z funkcji **MOTORS_go_*** lub **MOTORS_rotate** wywoływane są odpowiednio funkcje **MOTORS_M*_forward** lub **MOTORS_M*_backward**, ewentualnie zmieniane są wartości zadane przy poruszaniu się po przekątnych.

Nastawy regulatora PID dobierane były metodą empiryczną. Dobry efekt uzyskałam przy nastawach P = 1.0, I = 20.0, D = 0.0, zatem regulator był regulatorem PI. Przy późniejszym rozwoju projektu zostaną przetestowane inne kombinacje, jednak ta dawała zadowalające rezultaty.



Rysunek 20: Sposób sterowania robotem z kołami mecanum. Źródło: [15].

8 Podsumowanie

Mimo problemów z przystosowaniem czujnika odległości do otrzymywania poprawnych danych, projekt można uznać za zakończony z sukcesem. Robot spełnia swoje zadania (co można zobaczyć, oglądając filmik dostępny pod linkiem `https://youtu.be/BOfTCmvky2o`). Cały projekt znajduje się na dwóch repozytoriach: jednym z kodem do aplikacji (`https://github.com/emilia-szymanska/waddles_app`) i drugim z kodem do projektu STM32 (`https://github.com/emilia-szymanska/waddles_stm32/`).

Literatura

[1] Wikipedia, "Koła mecanum." https://en.wikipedia.org/wiki/Mecanum$_w heel$.

[2] S. L. Dickerson and B. D. Lapin, "Control of an omni-directional robotic vehicle with mecanum wheels," in *NTC '91 - National Telesystems Conference Proceedings*, pp. 323–328, 1991.

[3] P. Papcun, I. Zolotova, and K. Tafsi, "Control and teleoperation of robot khepera via android mobile device through bluetooth and wifi," *IFAC-PapersOnLine*, vol. 49, no. 25, pp. 188–193, 2016. 14th IFAC Conference on Programmable Devices and Embedded Systems PDES 2016.

[4] STMicroelectronics, "STM32F411xC STM32F411xE Datasheet." December 2017.

[5] Components101, "HC Serial Bluetooth Products User Instructional Manual." 2018.

[6] A. M. Hobby, "Moduł Bluetooth Slave HC06." http://sklep5296580.homesklep.pl/pl/p/Modul-Bluetooth-Slave-HC06-Arduino-HC-06-AVR/1836.

[7] I. sklep elektroniczny Botland, "Silnik HPCB 150:1 obustronny wał - Pololu 3076." https://botland.com.pl/.

[8] I. sklep elektroniczny Botland, "Enkodery magnetyczne do micro silników Pololu 2,7-18V - Pololu 3081." https://botland.com.pl/sterowniki-silnikow-dc/851-drv8835-dwukanalowy-sterownik-silnikow-11v-12a-pololu-2135.html.

[9] T. Instruments, "DUAL LOW VOLTAGE H-BRIDGE IC DRV8835 Datasheet." March 2012.

[10] I. sklep elektroniczny Botland, "DRV8835 - dwukanałowy sterownik silników 11V/1,2A - Pololu 2135." https://botland.com.pl/sterowniki-silnikow-dc/851-drv8835-dwukanalowy-sterownik-silnikow-11v-12a-pololu-2135.html.

[11] M. T. Inc., "MCP9808 datasheet." 2011.

[12] L. Adafruit, "Czujnik temperatury wysokiej precyzji MCP980." https://learn.adafruit.com/adafruit-mcp9808-precision-i2c-temperature-sensor-guide.

[13] E. J. Morgan, "Ultrasonic Ranging Module HC - SR04 datasheet." Nov. 16 2014.

[14] P. M. Projects, "Ultradźwiękowy czujnik odległości HCSR-04." http://ccspicc.blogspot.com/search/label/HC-SR04.

[15] Servomagazine, "Get rolling with omni-directional wheels." https://www.servomagazine.com/magazine/article/get-rolling-with-omni-directional-wheels.