Projekt

Roboty mobilne 1

Raport końcowy

Sterowanie holonomicznym robotem mobilnym "Waddles"

Skład grupy:
Emilia Szymańska, 248975

Termin: czwTP18

Prowadzący:
mgr inż. Arkadiusz Mielczarek

6 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 z Robotów Mobilnych 1 główny nacisk zostanie postawiony na stworzenie aplikacji komunikującej się z robotem, sposób nawiązywania połączenia między urządzeniami oraz sterowanie silnikami adekwatne do poleceń wysyłanych przez aplikację.

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 [3]) wymaga odpowiedniego skoordynowania kierunku i prędkości obrotu wszystkich czterech kół. Mobilne urządzenie stanowi lekką i poręczną alternatywę dla kontrolera, dlatego na wzór [8] planowane jest wykorzystanie komunikacji po Bluetooth z aplikacją na systemie Android.

Rysunek 1: Przykładowy robot wykorzystujący koła mecanum; źródło - [14]

1.1 Specyfikacja sprzętowa i zadaniowa

Do wykonania części elektronicznej robota niezbędne będą:

- mikrokontroler STM32 typu Black Pill,
- 4 silniki prądu stałego z enkoderami magnetycznymi wraz ze sterownikami,
- ultradźwiękowy czujnik odległości,
- czujnik temperatury,
- moduł Bluetooth,
- układ zasilania.

Czujnik temperatury odpowiedzialny będzie za monitorowanie temperatury w pobliżu mikrokontrolera i odcięcie zasilania w przypadku przegrzewania się układu. Czujnik odległości (w pierwotnej wersji zamieszczony z przodu pojazdu) będzie miał za zadanie zablokowanie wykonywania poleceń jazdy do przodu w przypadku wykrycia zbyt bliskiej przeszkody.

Urządzenie mobilne z zainstalowaną aplikacją powinno umożliwić użytkownikowi nawiązanie połączenia poprzez Bluetooth z pojazdem i wybór kierunku jazdy za pomocą odpowiednich przycisków, po naciśnięciu których adekwatna komenda jest wysyłana przez aplikację.

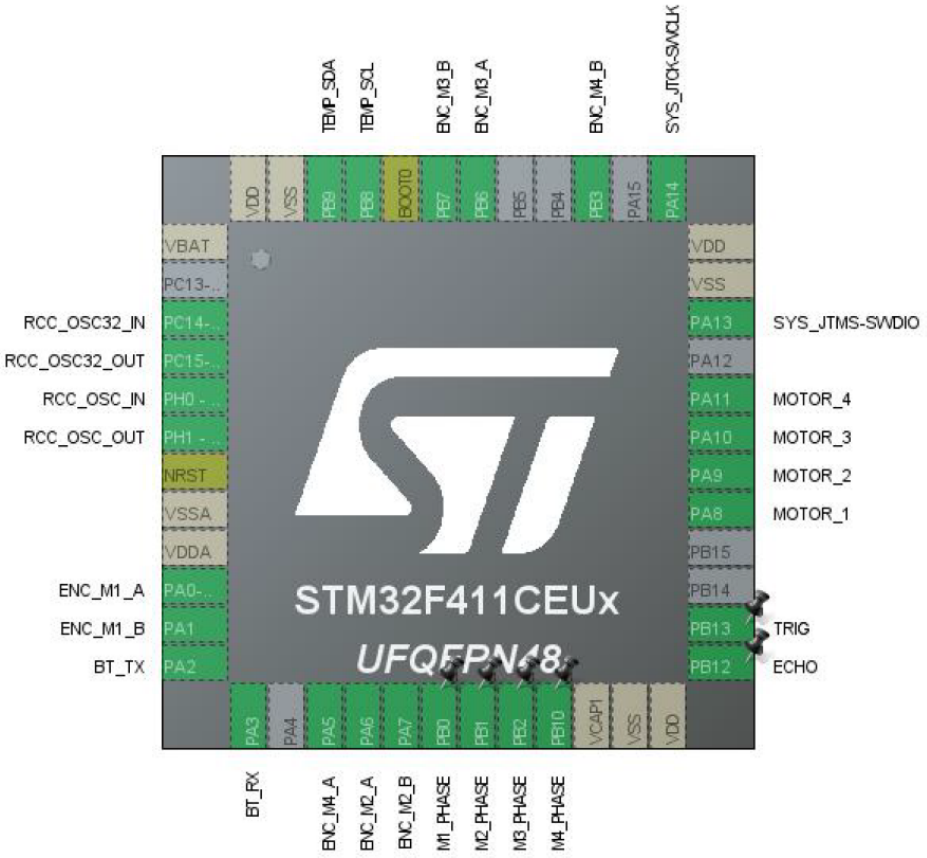
Obudowa na część elektroniczną oraz same koła mecanum (w pierwotnym założeniu) zostaną wydrukowane na drukarce 3D, zatem filament Easy PLA również jest potrzebny, jednak niewykluczone są zmiany w tym zakresie po wykonaniu pierwszych testów.

2 Harmonogram

1. Zakupienie wszystkich niezbędnych komponentów i materiałów, przygotowanie podstawowej wersji aplikacji - 21.03.
2. Podłączenie czujnika temperatury i czujnika odległości do mikrokontrolera, stworzenie programu sczytującego dane z czujników - 26.03.
3. Podłączenie modułu Bluetooth, napisanie programu obsługującego ten moduł - 2.04.
4. Podłączenie enkoderów i silników ze sterownikami do mikrokontrolera, nauka odbierania danych i generowania sygnałów na zegarach - 16.04.
5. Rozbudowa aplikacji o wysyłanie komend do pojazdu, zaimplementowanie obsługi komend na mikrokontrolerze - 30.04.
6. Zaprojektowanie, wydrukowanie i złożenie obudowy wraz z kołami - 14.05.
7. Zaprojektowanie i implementacja układu zasilania - 21.05.
8. Zintegrowanie całego systemu, nanoszenie ewentualnych poprawek - 04.06.

3 Konfiguracja STM32

Na rys. 2 została przedstawiona konfiguracja użytych pinów w mikrokontrolerze. Wykorzystane zostały piny GPIO (czujnik odległości), TIM (zegar podstawy czasu, sygnały PWM, odczytywanie danych z enkoderów), I2C (czujnik temperatury) oraz USART (moduł Bluetooth).



Rysunek 2: Konfiguracja wyjść mikrokontrolera w programie STM32CubeMX

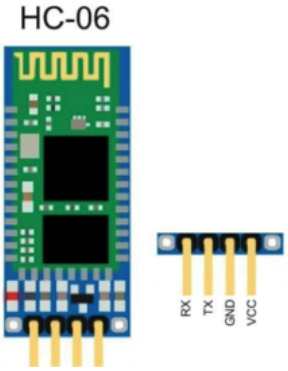
4 Urządzenia zewnętrzne

4.1 Moduł Bluetooth

Użyty moduł (HC-06) [2] 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.3). Nie można wywoływać komend AT po połączeniu się z urządzeniem przy pomocy Bluetootha, dlatego konwerter był konieczny, by móc cokolwiek zmienić.

Rysunek 3: Widoczne urządzenie z perspektywy komputera

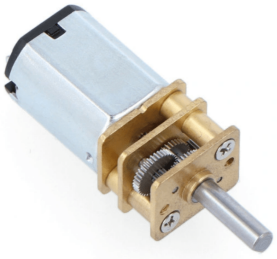
Prezentacja modułu znajduje się na Rys.4. Wymaga on połączenia pinów RX do BT_TX na STM32 oraz TX do BT_RX na STM32. Podane zasilanie było na poziomie 3V3.



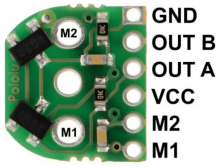
Rysunek 4: Moduł Bluetooth HC06. Źródło: [4].

4.2 Sterowniki silników, enkodery i silniki

Mikro silniki prądu stałego firmy Pololu z przekładnią 150:1 oraz prędkością obrotową 200 obr/min (Rys.5) zostały połączone z enkoderami magnetycznymi (Rys.6) tego samego producenta. Silniki zasilone zostały napięciem 8V, logika enkoderów natomiast napięciem 3V3.

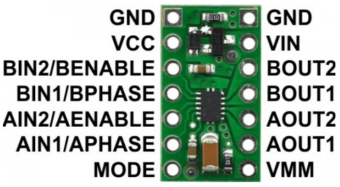


Rysunek 5: Silniki Pololu HPCB. Źródło: [13].



Rysunek 6: Enkodery magnetyczne Pololu. Źródło: [12].

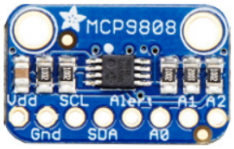
Nad sterowaniem silnikami czuwa dwukanałowy sterownik silników DRV8835 ([6], Rys. 7). Sterując sygnałem PWM (piny ENABLE połączone z MOTOR_x) oraz podając odpowiednie wyjście na GPIO (piny PHASE połączone z Mx_PHASE) można sterować prędkością obrotową wału oraz kierunkiem obrotu dwóch silników. Należy 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 7: Dwukanałowy sterownik silników DRV8835 Pololu. Źródło: [11].

4.3 Czujnik temperatury

Czujnik temperatury MCP9808 firmy Adafruit (Rys. 8, [5]) 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.

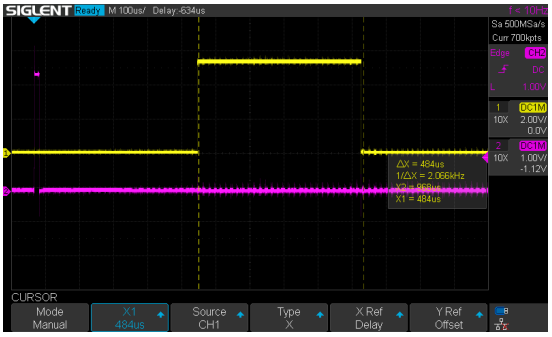


Rysunek 8: Czujnik temperatury wysokiej precyzji MCP9808. Źródło: [1].

4.4 Czujnik odległości

Użyty czujnik odległości HCSR-04 ([7], Rys. 9) 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. 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 9: Ultradźwiękowy czujnik odległości HCSR-04. Źródło: [9].



(a) Inicjowanie odczytu.

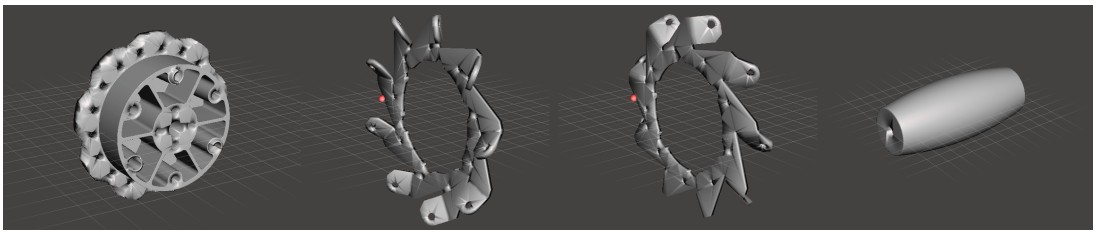
(b) Zakłócenia sygnału.

Rysunek 10: Odczyty sygnałów na oscyloskopie związanych z czujnikiem odległości.

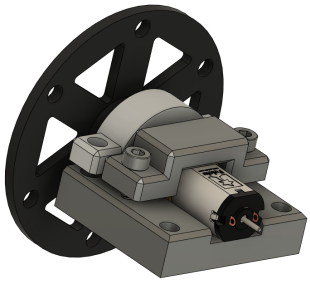
Czujnik ultradźwiękowy potrafi jednak zwracać co jakiś czas losowe wartości, co widać na rysunku 10b - mimo braku zmiany odległości część sygnałów ECHO jest krótsza, a część dłuższa. W związku z tym do analizy odległości wykorzystuję średnią z ostatnich ośmiu próbek.

5 Konstrukcja mechaniczna

Modele kół oraz elementów przytrzymujących silniki zostały zaczerpnięte z portalu Thingiverse - zdjęcia modeli zostały przedstawione odpowiednio na rys. 11 i 12.



Rysunek 11: Modele kół mecanum.

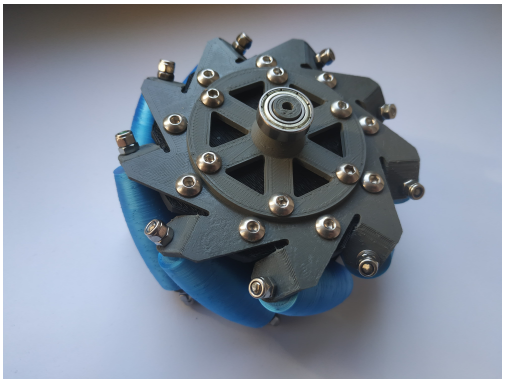


Rysunek 12: Modele elementów przytrzymujących silniki.

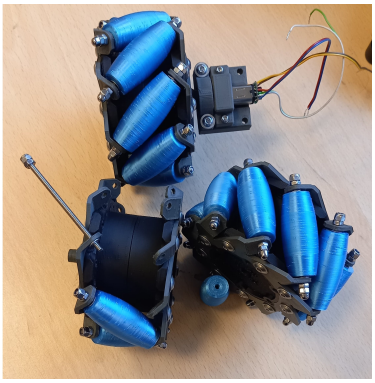
Model wymagał wydrukowania na drukarce 3D, powiercenia otworów oraz rolek i gwintowania niektórych otworów. Do złożenia kół potrzebne były łożyska, śruby (M3 i M4) z łbem walcowym, nakrętki, podkładki (w tym też sprężyste) i pręt gwintowany (później pocięty na mniejsze części). Proces składania kół widoczny jest na rys. 13.

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. 14.

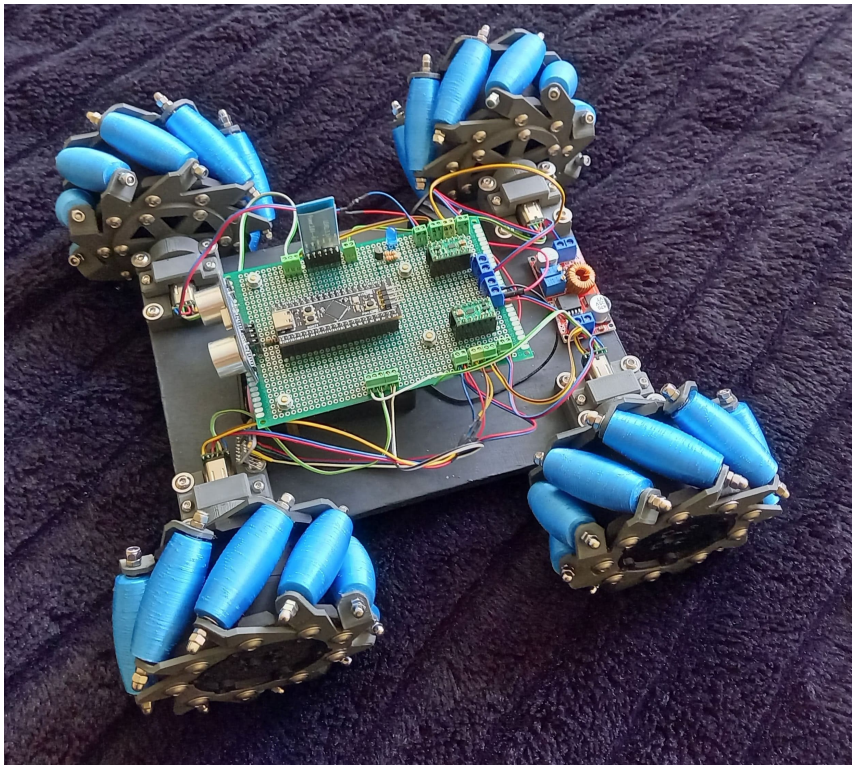


(a) Złożone koło.



(b) Montaż kół.

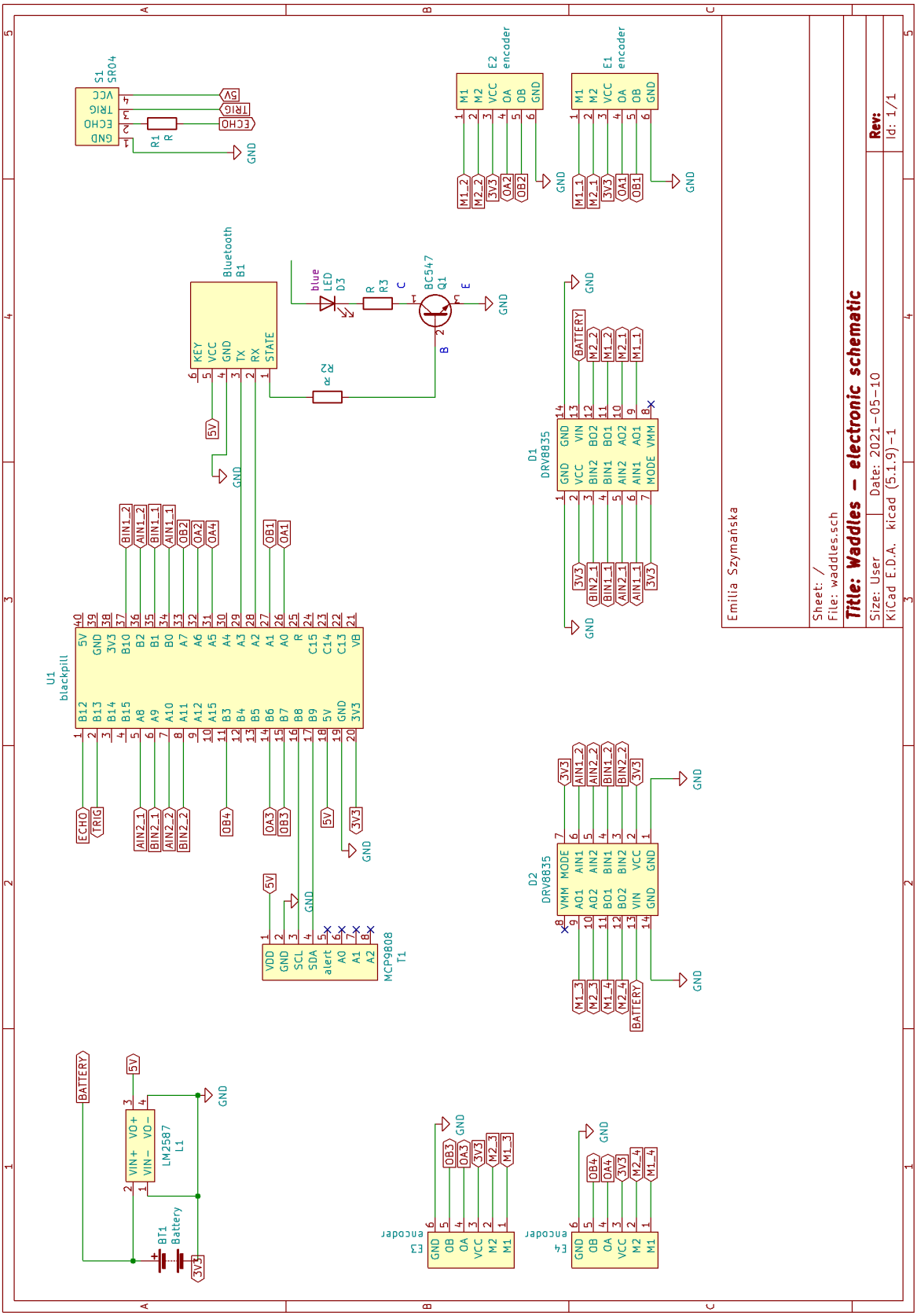
Rysunek 13: Gotowe koła mecanum.



Rysunek 14: Złożony robot.

6 Projekt elektroniki

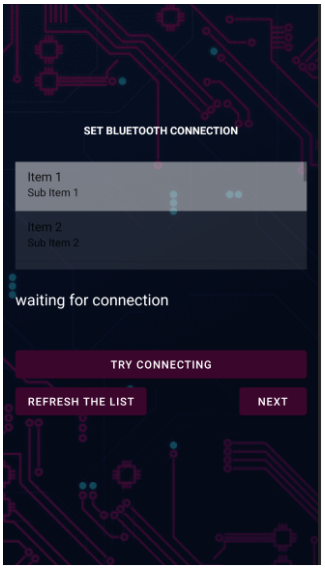
Schemat elektroniki i połączeń przedstawiony został na rys. 15.



Rysunek 15: Schemat elektroniczny robota.

7 Aplikacja mobilna

Aplikacja mobilna została napisana z przeznaczeniem na system Android. Użytym językiem programowania jest Java z odpowiednim API do Androida. Kod aplikacji można znaleźć na repozytorium `https://github.com/emilia-szymanska/waddles_app`



(a) Układ okna głównego.

(b) Układ okna ze strzałkami.

Rysunek 16: Layout aplikacji mobilnej.

7.1 MainActivity.java

Główna klasa aplikacji *MainActivity.java* odpowiada układowi opisanemu w pliku *main_ activity.xml*. Efekt takiego układu jest widoczny na rys. 16a. Okno zawiera w sobie listę sparowanych wcześniej przez Bluetooth urządzeń, pole tekstowe informujące o stanie połączenia oraz trzy przyciski. Po wybraniu z listy odpowiedniego urządzenia należy wybrać *TRY CONNECTING*. Jeśli połączenie się powiedzie, zostanie załączony przycisk *NEXT* umożliwiający przejście do drugiego okna. W przeciwnym razie pokaże się odpowiedni komunikat i ponownie można wybrać urządzenie. W każdym momencie można odświeżyć listę urządzeń przyciskiem *REFRESH THE LIST*.

7.2 ArrowActivity.java

Klasa *ArrowActivity.java* odpowiada układowi opisanemu w pliku *arrow_ activity.xml* (rys. 16b) i jest oknem dostępnym po przejściu z głównego okna. Klasa ta zawiera w sobie dziesięć przycisków. Jedno z nich pozwala na cofnięcie się do wyboru połączenia z danym urządzeniem, pozostałe dziewięć umożliwia wysyłanie odpowiednich komend za pomocą Bluetooth. Przy wybraniu danego przycisku pole *message* klasy *ArrowActivity.java* zmienia swoją wartość na odpowiadającą danemu kierunkowi. Przy puszczeniu przycisku pole to również zmienia swoją wartość, jednak na domyślną *nn*. Co 50ms osobny wątek wysyła przez połączenie Bluetooth wartość pola *message*, dzięki czemu główny wątek programu nie jest przeciążony.

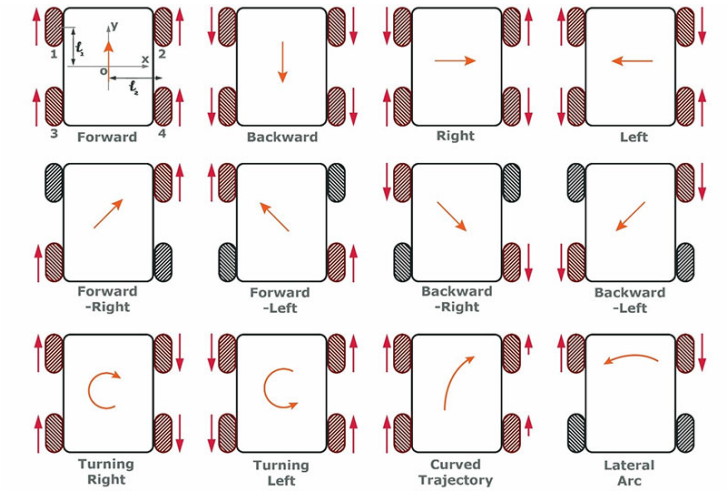
7.3 Komendy

Jak zostało to wspomniane, wybranie odpowiedniego przycisku powoduje wysłanie odpowiedniej komendy. Dla ułatwienia, komenda ta zawsze ma dwa znaki i interpretowana jest w następujący sposób:

- **nn** - silniki mają się zatrzymać (wszystkie wypełnienia PWM podane są jako 0),
- **ff** - jazda do przodu,
- **bb** - jazda do tyłu,
- **rr** - jazda w prawo,
- **ll** - jazda w lewo,
- **fr** - jazda na ukos (w przód i w prawo),
- **fl** - jazda na ukos (w przód i w lewo),
- **br** - jazda na ukos (w tył i w prawo),
- **bl** - jazda na ukos (w tył i w lewo),
- **cc** - obrót wokół własnej osi zgodnie z ruchem wskazówek zegara.

Sposób sterowania silnikami jest przedstawiony na rys. 17. 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 17: Sposób sterowania robotem z kołami mecanum. Źródło: [10].

8 Opis działania programu na STM32

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

1. 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).

2. 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).

3. 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).

4. 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).

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.

8.1 Główna pętla programu

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

8.1.1 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.

```
1  while (1)
2    {
3      .....
4      if(command_flag)
5      {
6        if(strcmp(command, "nn") == 0)
7        {
8          setpoints_to_zeros(setpoints);
9        }
10       else if(strcmp(command, "ll") == 0)
11       {
12           setpoints_to_const(setpoints);
13           MOTORS_go_left();
14       }
15       else if(strcmp(command, "rr") == 0)
16       {
17           setpoints_to_const(setpoints);
18           MOTORS_go_right();
19       }
20       else if(strcmp(command, "bb") == 0)
21       {
22           setpoints_to_const(setpoints);
23           MOTORS_go_backward();
24       }
25       else if(strcmp(command, "ff") == 0)
26       {
27         if(average_distance <= DISTANCE_LIMIT)
28         {
29             setpoints_to_zeros(setpoints);
30         }
31         else
32         {
33           setpoints_to_const(setpoints);
34           MOTORS_go_forward();
35         }
36       }
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    }
```

8.1.2 Odczyt temperatury i odległości

Pierwsza sekcja (zależna od flagi *main_flag* wystawianej od przerwania licznika TIM10 z częstotliwością 10 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. Powód takiego obliczania odległości, jaki jest zaprezentowany poniżej, wytłumaczony jest w sekcji dotyczącej czujnika ultradźwiękowego.

```
1  while (1)
2    {
3      .....
4      if(main_flag)
5      {
6        temperature = TEMP_get_data();
7        if(temperature >= TEMPERATURE_ALERT)
8        {
9            setpoints_to_zeros(setpoints);
10         temperature_alert();
11       }
12       distance = USONIC_get_distance();
13       distance_sum = distance_sum - distance_array[current_index] +  distance;
14       distance_array[current_index] = distance;
15       current_index = (current_index + 1) % measurements;
16       average_distance = distance_sum / measurements;
17       main_flag = 0;
18     }
19     .....
20   }
```

8.1.3 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.

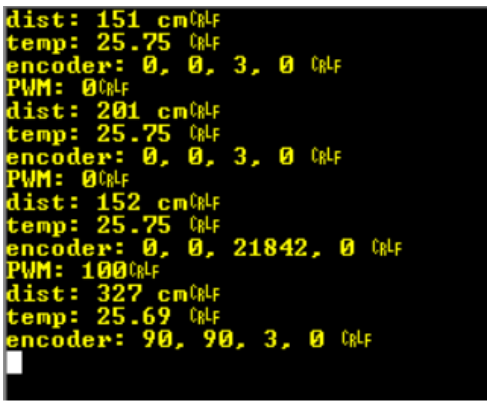
```
while (1)
  {
    .....
    if(control_flag)
    {
      ENCODERS_get_rpm();
      MOTORS_set_speed(1, PID_output(&pid_m1, setpoints[0], E1_rpm));
      MOTORS_set_speed(2, PID_output(&pid_m2, setpoints[1], E2_rpm));
      MOTORS_set_speed(3, PID_output(&pid_m3, setpoints[2], E3_rpm));
      MOTORS_set_speed(4, PID_output(&pid_m4, setpoints[3], E4_rpm));
      control_flag = 0;
    }
    .....
  }
```

8.2 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.

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

[2] Components101. HC Serial Bluetooth Products User Instructional Manual. 2018.

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

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

[5] M. T. Inc. MCP9808 datasheet. 2011.

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

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

[8] P. Papcun, I. Zolotova, K. Tafsi. Control and teleoperation of robot khepera via android mobile device through bluetooth and wifi. *IFAC-PapersOnLine*, 49(25):188–193, 2016. 14th IFAC Conference on Programmable Devices and Embedded Systems PDES 2016.

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

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

[11] 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.

[12] 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.

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

[14] Wikipedia. Koła mecanum. https://en.wikipedia.org/wiki/Mecanum_wheel.