AKADEMIA GÓRNICZO-HUTNICZA IM. STANISŁAWA STASZICA W KRAKOWIE

WYDZIAŁ INŻYNIERII METALI I INFORMATYKI PRZEMYSŁOWEJ

KATEDRA INFORMATYKI STOSOWANEJ I MODELOWANIA

Praca dyplomowa inżynierska

„Konstrukcja oraz implementacja zdalnie sterowanego robota przeznaczonego do wykonywania podstawowych pomiarów atmosferycznych”

„Design and implementation of a remotely controlled robot designed to perform basic atmospheric measurements”

Autor: *Mateusz Piotr Stabryła*
Kierunek studiów: Informatyka Stosowana
Opiekun pracy: *dr inż. Piotr Kustra*

Kraków, 2021

Spis treści

1. Wstęp

W ramach pracy inżynierskiej zrealizowano projekt, który ma na celu stworzenie produktu o wielu różnorodnych zastosowaniach w zakresie dokonywania pomiarów. Produktu, który nie byłby uzależniony od zastosowania tylko w jednej dziedzinie, ale byłby łatwo modyfikowalnym rozwiązaniem, które można by było dostosowywać do wielu różnych potrzeb. W tym celu skonstruowano prototyp robota z zestawem kilku czujników atmosferycznych, silnikami do poruszania się oraz modułem do komunikacji WiFi. Idea tej pracy polega na tym, aby szeroka gama klientów o różnych potrzebach i budżecie mogła odnaleźć zastosowanie dla zaprezentowanego niżej rozwiązania i wynieść z niego wymierne korzyści. Oczywiście trudno jest stworzyć projekt wielofunkcyjny, który mógłby konkurować we wszystkich płaszczyznach z innymi dokładniejszymi narzędziami w danych dziedzinach np. w kwestii dokładnego pomiaru temperatury czy też innych parametrów atmosferycznych. Jednakże możliwe jest stworzenie produktu, który może łączyć kilka funkcjonalności (przykładowo możliwość autonomicznego poruszania się oraz zbierania danych z czujników pomiarowych), dzięki którym można dane operacje zrealizować z bezpiecznej odległości lub nawet bez bezpośredniego udziału człowieka.

Do tego projektu został dobrany odpowiedni szkielet prototypu z istniejącego już robota. Robot o którym mowa, to Miabot PRO BT v2, jeden z elementów robotycznego zestawu do gry w piłkę nożną . Roboty należące do tego zestawu posiadały silniki DC oraz kontroler sterujący ruchem. Ze względu na uszkodzenia, szkielet jednego z robotów z tego zestawu został zmodernizowany i wykorzystany do implementacji prototypu.

Robot w celu zrealizowania założeń, które zostały postawione w tym projekcie, musiał on być pozbawiony poprzednich części (głównie kontrolera mikrorobota) . Na potrzeby prototypu większość instalacji elektronicznych przebiega poza obudową.

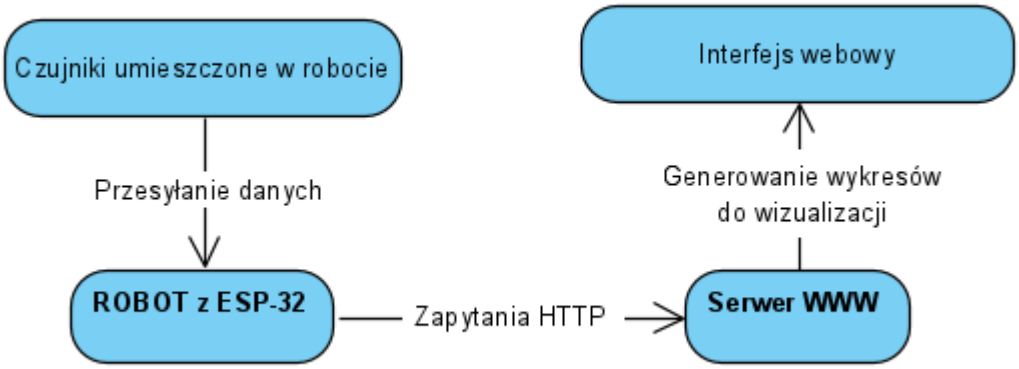
W ramach tej pracy udało się nawiązać współpracę z inną studentką - Aleksandrą Ćwikłą, studentką wydziału Inżynierii Metali i Informatyki Przemysłowej . Jej zadaniem w ramach pracy magisterskiej było stworzenie aplikacji internetowej pozwalającej agregować i przetwarzać dane zbierane przez dowolną ilość czujników za pomocą komunikacji http. Ze względu na zbieżność celów obu projektów przeprowadzono integrację obu rozwiązań, gdzie robot przyjmuje rolę dostarczyciela danych dla systemu.

2. Cel pracy

Podczas poszukiwania podobnych rozwiązań, udało się odnaleźć przykład zastosowania podobnego projektu. Praca pod tytułem „Efficient Measurement Planning for Remote Gas Sensing with Mobile Robots" [26] jest dziełem 6 inżynierów z uniwersytetu Örebro w Szwecji. Praca opisuje sposób opracowania mobilnej platformy do wykrywania gazów w trudnych warunkach między innymi w przestrzeni otwartej. W tej pracy zwrócono uwagę na to, że klasyczne wykrywacze stacjonarne są dużo tańszym rozwiązaniem w porównaniu do wszelkim mobilnych wykrywaczy [27], aczkolwiek w przypadku wykrywa zagrożeń dla środowiska pomiary stacjonarne wymagałyby ogromnej ilości czujników. Dodatkowym atutem mobilnego rozwiązania jest technologia TDLAS [28], która pozwalałaby na wykrywanie potencjalnego zagrożenia w odległości bezpiecznej nie tylko dla człowieka sterującego mobilnym urządzeniem, ale także dal samego urządzenia

Celem tej pracy było skonstruowanie oraz oprogramowanie autonomicznego robota posiadającego napęd dwukołowy z silnikami DC oraz zestawem czujników monitorujących podstawowe parametry atmosferyczne. Robot ten ma na celu wykonywanie pomiarów z różnych miejsc z danego pomieszczenia zamkniętego oraz przesyłanie ich do aplikacji internetowej za pomocą interfejsu WiFi. Dzięki oprogramowaniu mikrokontrolera, robot jest w stanie samodzielnie poruszać się w tzw. trybie wall-following oraz w trybie sterowania ręcznego przez użytkownika.

Skonstruowany robot miałby być jedynie częścią większego rozwiązania (zobrazowanego na Rysunku nr 1), jaki planowałem przy okazji realizacji tej pracy inżynierskiej. Tym rozwiązaniem byłby zestaw do wykonywania pomiarów atmosferycznych w różnorodnych warunkach. Zestaw ten zawierałby w sobie kilkadziesiąt robotów opartych na prototypie, które miałby za zadanie równolegle zbierać wszystkie możliwe do odczytania dane z różnych miejsc danego pomieszczenia. Roboty byłyby zarządzanie przez centralny serwer WiFi przypisany do przenośnej stacji pozwalającej objąć swoim zasięgiem obszar o promieniu do 200m [1].



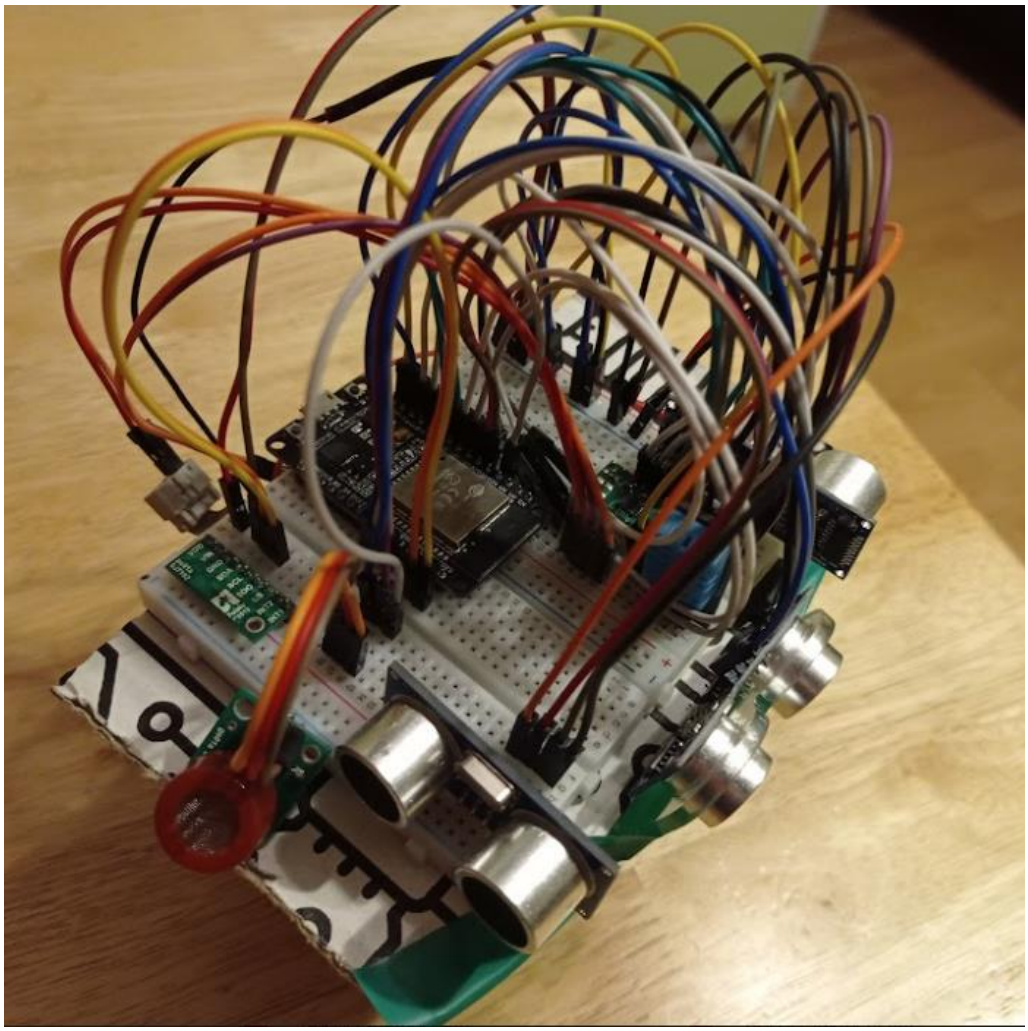
Rys. 1 - Ogólny schemat działania projektu

Tego rodzaju zestaw mógłby być prezentowany oraz sprzedawany jako mobilne narzędzie do realizacji pomiarów w miejscach o dużej powierzchni bez odpowiedniej infrastruktury lub w miejscach niebezpiecznych ze względu na skażenie chemiczne lub biologiczne. Dzięki modularności robotów, każda osoba mogłaby dostosować zestaw robotów pod własne potrzeby. Biorąc pod uwagę istniejące już pomysły [26] na podobne projekty robot miały pomóc w dokładniejszym wykrywaniu zagrożeń, szczególnie w miejscach niebezpiecznych dla człowieka (jak na przykład miejsce skażone zanieczyszczeniami).

3. Konstrukcja robota

Aktualny projekt jest modernizacją projektu starszego robota z zestawu do piłki nożnej. Jest to robot postaci bliskiej sześcianu o długości boku wynoszącej 7,50 centymetrów. Wewnątrz znajdują się dwa silniki prądu stałego (silniki DC) przyjmujące prąd o napięciu 6 Volt , które napędzają dwa koła o średnicy 5 centymetrów. Na ilustracji nr 2 i 3 jest pokazany skonstruowany prototyp.

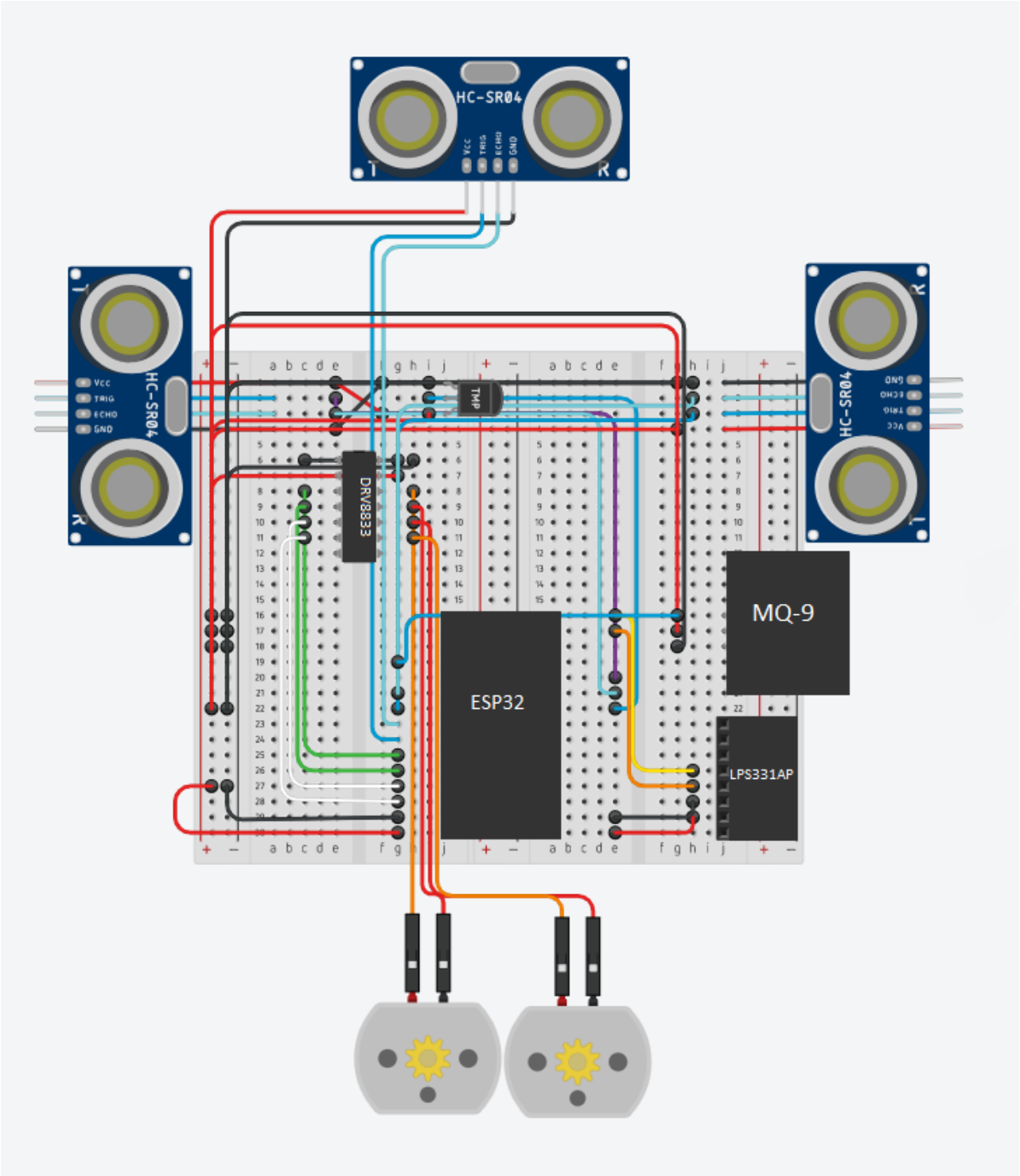
Na obudowie umieszczona została cała instalacja elektroniczna sterującą robotem. Składa się ona z rozszerzonej płytki stykowej, mikrokontrolera ESP32[2] oraz zestawu czujników przymocowanych do płytki. Silniki są połączone poprzez wyjście z tyłu robota. Dodatkowo wewnątrz obudowy robota znajduje się zestaw 4 baterii alkaicznych, które zasilają instalację napięciem 6V.



Rys. 2 - Robot autonomiczny - widok z góry, źródło: opracowanie własne

Rys. 3 - Robot autonomiczny - widok z boku, źródło: opracowanie własne

Ze względów na zbyt mały rozmiar oryginalnego robota cała elektronika wraz z okablowaniem musiała zostać poprowadzona na zewnątrz obudowy. Na rysunku 4 zaprezentowano schemat połączenia przewodów oraz komponentów na płytce stykowej.

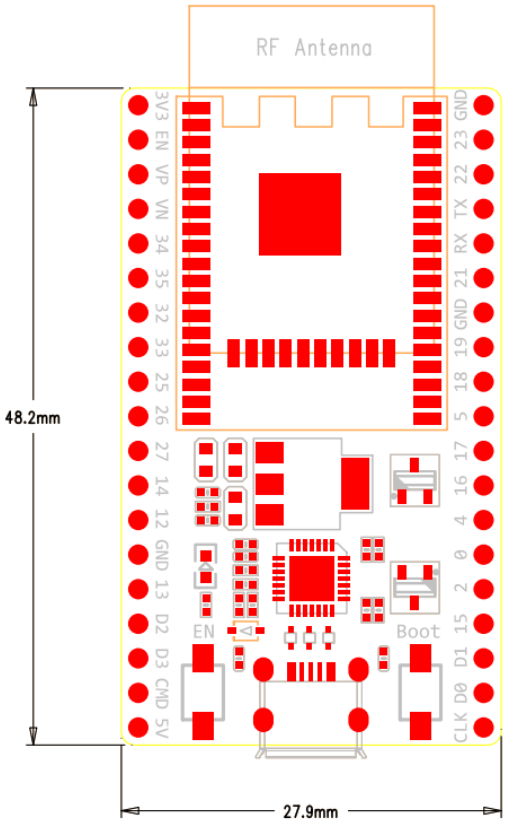


Rys. 4 - Schemat układu, źródło: opracowanie własne na podstawie programu Tinkercad [5]

3.1 Mikrokontroler ESP32

Do sterowania robotem został wykorzystany mikrokontroler ESP32-Devkit z rodziny modułów ESP-WROOM-32 z dodatkowym modułem WiFi oraz Bluetooth (pokazany na rysunku nr 5). Wybór tego rodzaju mikrokontrolera zależał od kilku czynników. Powyższy model, poza 30 wyjściowymi pinami umieszonymi w module (opisanymi na rysunku nr 6), posiada moduły do komunikacji w standardzie WiFi 2,4 GHz oraz Bluetooth BLE / v4,2. W szczególności brano pod uwagę możliwość komunikacji sieciowej, która pozwalałaby w wygodny sposób komunikować się z serwerem zarządzającym. Powyżej opisany mikrokontroler został wybrany przede wszystkim ze względu na małe gabaryty i możliwość sprawnej komunikacji sieciowej.

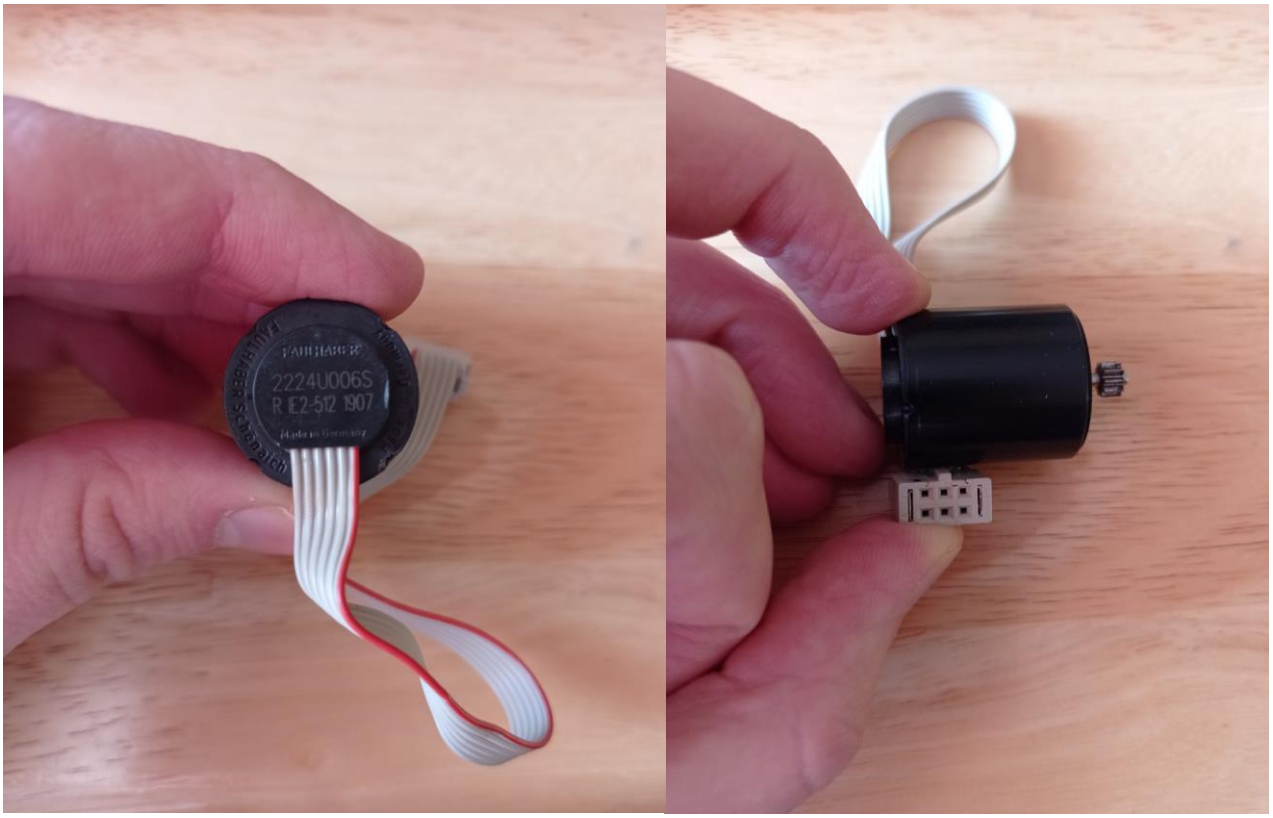
Rys. 5 - Mikrokontroler ESP32-Devkitm źródło: Opis ESP32 [3]



Rys. 6 - Schemat modułu ESP32-WiFi, źródło: Dokumentacja techniczna ESP32-Devkit [4]

3.2 Silniki DC

Wewnątrz obudowy robota znajdują się dwa silniki prądu stałego o numerze 2224U006SR (rysunek nr 7.1). Silniki są zasilane napięciem do 6V i napędzają koła o średnicy 5 centymetrów. Każdy z silników ma 6-pinowe wejście przeznaczone na zasilanie (pokazane na rysunku 7.2) oraz kalibrację prędkości kół. W przypadku omawianej konstrukcji do poprawnej obsługi wystarczyły pierwsze (patrząc od oznaczonego czerwoną linią przewodu) dwa piny zasilające. Jednakże w celu kompleksowego sterowania robotem potrzebna jest możliwość zmiany kierunku obrotu silnika, tak aby była możliwa jazda do przodu oraz do tyłu. Kierunek obrotu kół jest zależny od miejsc wpięcia napięcia oraz masy, do dwóch pinów zasilających.

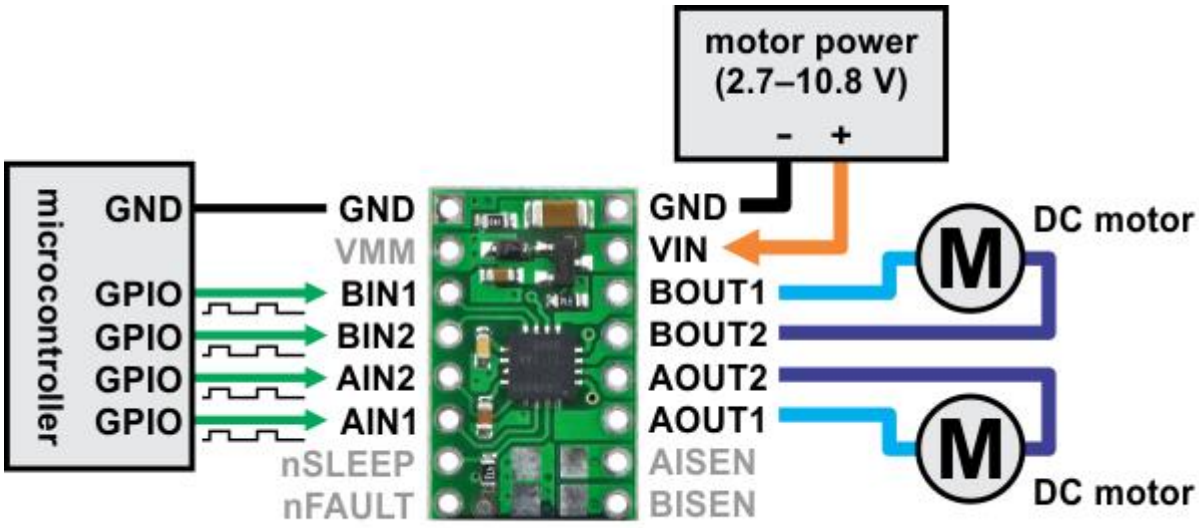


Rys. 7.1 - Silnik prądu stałego z etykietą; 7.2 - silnik i wejście 6-pinowe, źródło: opracowanie własne

Aby można było programowo sterować zachowaniem robota, potrzebne było użycie odpowiednich sterowników, które pozwalały by sterować prędkością (poprzez zmianę napięcia metodą PWM) oraz kierunkiem obrotu kół. W tym celu do układu wprowadzono odpowiedni sterownik.

3.2.1 Sterownik silników DC DRV8833

Sterownik ten pozwala na jednoczesne sterowanie dwoma silnikami prądu stałego wraz prędkością i kierunkiem. Posiada 16 pinów, w tym dwa analogowe wejścia sterujące typu PWM na każdy silnik DC. W zależności stanu pinów wejściowych sterownik generuje napięcie na pinach wyjściowych. Poza kierunkiem obrotu, sterownik umożliwia też sterowanie szybkością obrotu oraz sposobem zmiany prędkości .



Rys. 8 - Opis podpięcia układu sterownika DRV8833, źródło: opis ze strony producenta [6]

Zgodnie ze schematem podłączenia pokazanym na rysunku 8, sterownik posiada 4 wejścia oznaczone kolejno literą A i B, które symbolizują silnik przypisany do danego wejścia, oraz cyframi. Na każdy silnik przypadają 2 wejścia oraz wyjścia. Sygnał wchodzący możemy regulować za pomocą techniki PWM (*Pulse-Width Modulation).* Technika ta polega na ustawieniu częstotliwości zmiany stanu napięcia, dzięki czemu można ustalić długość trwania stanu wysokiego. Przekłada się ona zaś na moc idącą do silnika. W przypadku tego sterownika wypełnienie sygnału PWM na wejściu będzie zamieniana na napięcie idące do danego silnika DC. W poniższej tabelce możemy zobaczyć sposób, w jaki transformowany jest sygnał wejściowy dla jednego silnika.

Tabela 1 - Sposób kontroli mocy i kierunku silnika DC na podstawie stanów wejściowych, źródło: dokumentacja sterownika [7]

xIN1	xIN2	FUNCTION
PWM	0	FORWARD PWM, FAST DECAY
1	PWM	FORWARD PWM, SLOW DECAY
0	PWM	REVERSE PWM, FAST DECAY
PWM	1	REVERSE PWM, SLOW DECAY

W implementacji zastosowanej w tym robocie wykorzystany został sposób poruszania się w przód oraz do tyłu z metodą fast Decay (pozycja 1 i 3 w Tabelce nr 1). Pozwala ona na bardzo szybką zmianę prędkości, która jest bardzo potrzebna w małych i trudno dostępnych pomieszczeniach, w którym ma operować robot.

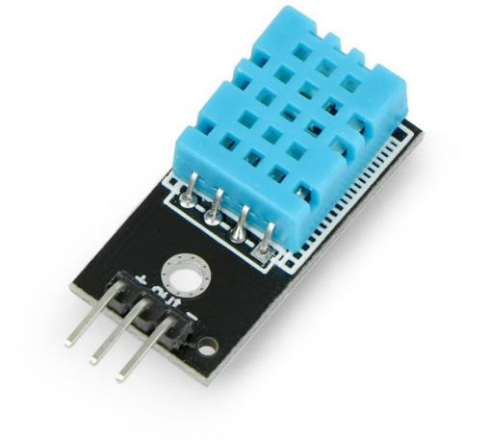
3.3 Czujniki

Głównym zadaniem robota jest pobieranie danych atmosferycznych i przesyłanie ich do serwera, który ma zadanie agregować te dane. W tym celu prototyp musiał zostać wyposażony w podstawowy zestaw czujników, który pozwoli na odczyt podstawowych informacji. Nie wszystkie dane pobierane z czujników są wykorzystywane do przesłania na serwer. Niektóre z tych danych, jak na przykład dane z czujników odległości, są wykorzystywane przez program mikrokontrolera do sterowania ruchem kół robota.

Ten prototyp posiada aktualnie niewielki zbiór czujników ze względu na niewielkie miejsce na obudowie oraz płytce prototypowej. Pobierają one podstawowe dane atmosferyczne zgodnie z wytycznymi do tego projektu.

3.3.1 Czujnik Temperatury i wilgotności DHT-11

Pierwszym czujnikiem umieszczonym na robocie jest czujnik temperatury i wilgotności DHT-11 (pokazany na rysunku nr 9). Składa się on się z małej płytki PCB – zawierającej rezystancyjny czujnik wilgotności i termistor. Przyjmuje on napięcie od 3,3 do 5 Volt. Potrafi on zmierzyć temperaturę w zakresie od 0°C do 50°C, a wilgotność od 20% do 90% RH (Relative Humidity).



Rys. 9 - Czujnik DHT-11, źródło: strona sprzedaży w sklepie Botland [8]

Czujnik wysyła dane poprzez pin określony jako „OUT”. W celu odczytania danych mikrokontroler musi wysłać sygnał stanu niskiego przez przynajmniej 18 ms i następnie rozpocząć odczyt sygnału. Czujnik wysyła dane w postaci 40 bitowych pakietów, który zawiera dane dotyczące aktualnej temperatury i wilgotności. [9]

3.3.2 Czujnik odległości HC-SR04

Ultradźwiękowy czujnik odległości HC-SR04 (pokazany na rysunku nr 10) służy przede wszystkim jako narzędzie dla robota, jako sposób orientacji w terenie. W robocie zostały umieszczone 3 czujniki tego typu: z przodu robota oraz po jego prawym i lewym boku.

Rys. 10 - Czujnik odległości HC-SR04, źródło: strona sprzedaży w sklepie Botland [10]

Czujnik jest zasilany napięciem 5V. Posiada zakres pomiarowy od 2 do 200 cm. Aby rozpocząć pomiar należy podać na pin TRIG impuls napięciowy (stan wysoki 5V) przez 10 us. Moduł dokonuje pomiaru odległości przy pomocy fali dźwiękowej o częstotliwości 40 kHz [10]. Do mikrokontrolera wysyłany jest sygnał, w którym odległość zależna jest od czasu trwania stanu wysokiego i można ją obliczyć ze wzoru:

$$dystans\,[cm] = (\,czasu\,stanu\,wysokiego\,[us] * 34\,) / 1000 / 2$$

Równanie nr 1 – wzór na odległość na podstawie czasu stanu wysokiego czujnika

Wzór ten można uprościć do postaci:

$$dystans\,[cm] = \frac{czasu\,stanu\,wysokiego\,[us]}{58}$$

Równanie nr 2 – uproszczony wzór z Równania nr 1 do postaci stosowanej w programie

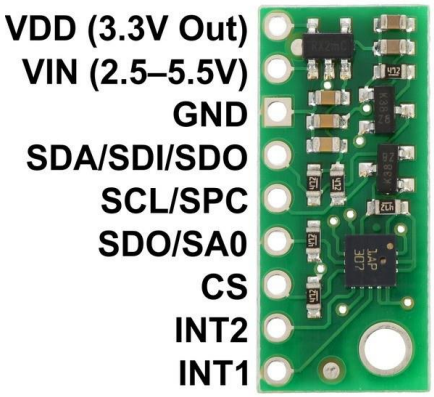
Dzięki temu równaniu można wyznaczyć odległość w zakresie od 2 do 200 cm. Na potrzeby poruszania się robota jest to wystarczający zakres.

3.3.3 Czujnik ciśnienia LPS331AP

Czujnik ciśnienia LPS331AP (pokazany na rysunku nr 11) to czujniki pozwalającym mierzyć jeden z podstawowych parametrów atmosfery jakim jest ciśnienie atmosferyczne, ale także pozwala na rozbudowaną orientację w terenie. Dzięki odpowiednim obliczeniom można wyznaczyć wysokość, na której znajduje się robot.

Czujnik przyjmuje napięcie w zakresie 2,5 do 5 Volt. Potrafi on mierzyć ciśnienie atmosferyczne w zakresie od 26 kPa do 126 kPa z dokładnością 0,2kPa.

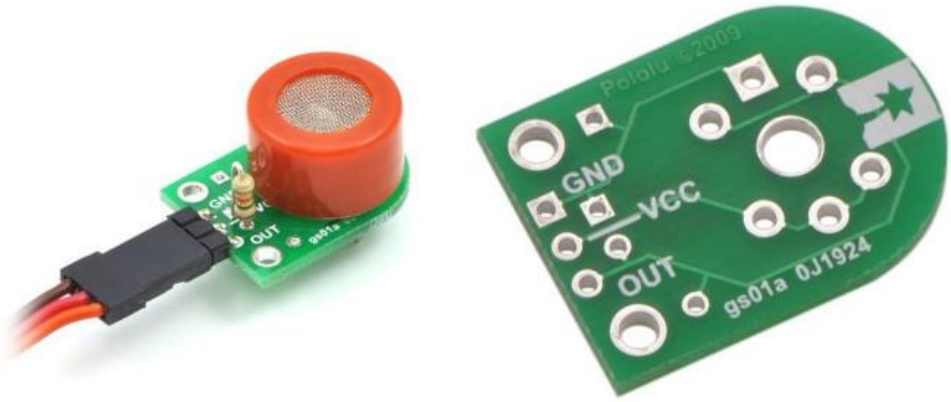
Komunikacja z czujnikiem odbywa się poprzez interfejs cyfrowy I2C. W celu wykorzystania tej magistrali podłączono mikrokontroler do wyprowadzeń czujnika SCL oraz SDA (zaznaczonymi na rysunku nr 11). Na mikrokontrolerze ustawiono dwa piny komunikacyjne, jako piny interfejsu I2C, aby odczyt danych z czujnika był możliwy.



Rys. 11 - Opis pinów na czujniku LPS331AP, źródło: strona sprzedaży w sklepie Botland [11]

3.3.4 Czujnik niebezpiecznych gazów MQ-9

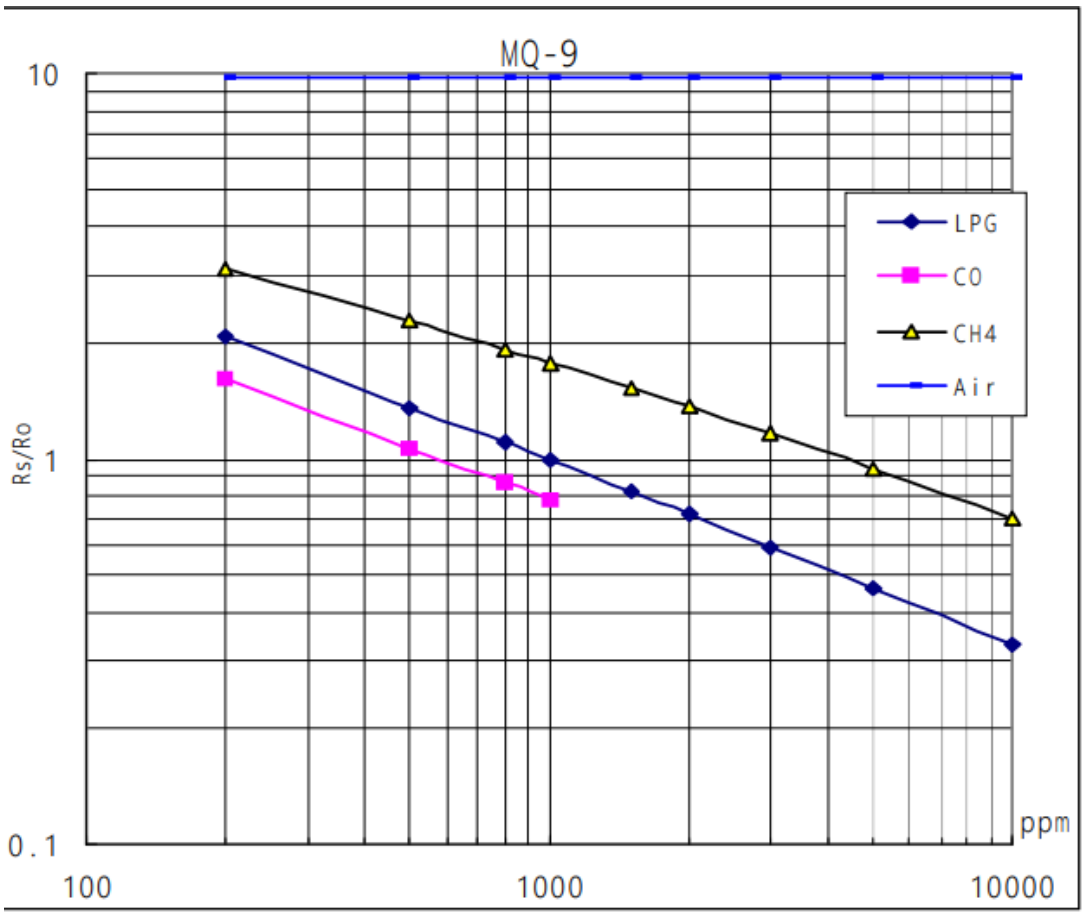
Czujnik ten jest jednym z rodzajów czujników z rodziny MQ. Czujniki te odczytują dane o szkodliwych gazach na podstawie przewodności przez specjalne materiały. W przypadku MQ-9 jest to tlenek SnO2. Dla czystego powietrza przewodność prądowa przez ten materiał jest minimalna, ale kiedy pojawi się odpowiedni gaz o trujących właściwościach, to przewodność wzrasta i sygnał wysyłany z czujnika jest wyraźniejszy.



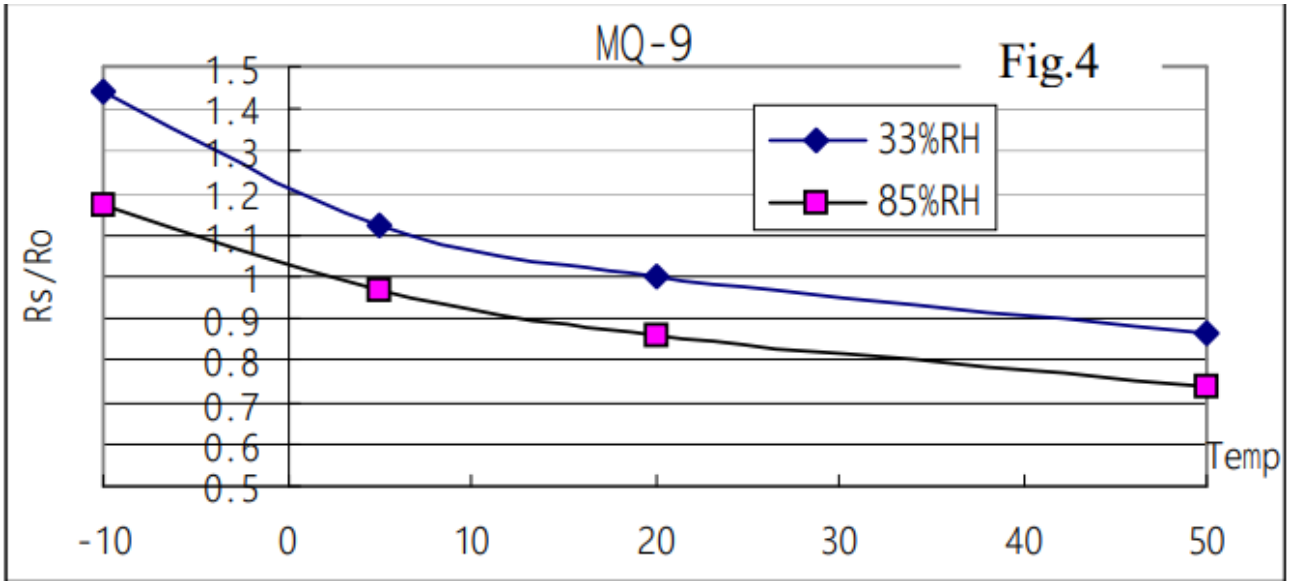
Rys. 12 - Czujnik MQ-9 na podstawce MQ - Pololu 1479, źródło: strona sprzedaży czujnika MQ-9 [12]

Czujnik MQ-9 (pokazany na rysunku nr 9) pobiera napięcie 5V. Wykrywa stężenie tlenku węgla oraz inny gazów trujących dla człowieka (głównie Metan i Butan). Wykrywanie gazu opiera się na badaniu rezystywności materiału, z którego użyto do czujnika. Zmierzoną rezystywność (opisaną na wykresie jako Rs) porównuje się z rezystywnością powietrza zawierającego 1000 ppm danego gazu (opisanej jako R0). Następnie oblicza się iloraz Rs/R0 i porównuje się wynik z wykresem nr 1.

Na wykresie nr 2 można też zauważyć zależność odbieranych wartości od wilgotności i temperatury.



Wykres 1 - Wykres zależności ilorazu Rs/R0 od stężenia danego gazu w powietrzu, źródło: dokumentacja MQ-9 [12]



Wykres 2 - Wykres zależności ilorazu Rs/R0 od wilgotności i temperatury, źródło: dokumentacja MQ-9 [12]

4. Implementacja oprogramowania

Do oprogramowania mikrokontrolera ESP-32 wykorzystano popularną technologię do oprogramowania mikroprocesorów Arduino IDE[13]. Jest to środowisko programistyczne wykorzystujące język C oraz wbudowane metody pozwalające zarządzać pinami wejściowymi oraz wyjściowymi. Arduino pozwalają na sprawną kompilację kodu i wysłanie go do mikroprocesora przez połączenie szeregowe. W przypadku mojej instalacji elektronicznej musiałem posłużyć się specjalną modyfikacją środowiska, a dokładniej musiałem dodać obsługę mikrokontrolera ESP-32[14] do środowiska, gdyż natywnie Arduino nie oferuje dla niego wsparcia.

Ze względu na to, że wiele funkcjonalności istniejących w klasycznym Arduino nie jest obsługiwany przez bibliotekę wystawioną na stronie repozytorium producenta ESP-32, dobrano kilka innych bibliotek w celu poprawnego wykorzystania między innymi generowania sygnału w technice PWM[15].

Samo oprogramowanie robota składa się z kilku istotnych elementów pozwalających mu realizować wcześniej założone zadania. Do podstawowych zadań należy:

1. Oprogramowanie sposobu poruszania się robotem
2. Pobierania oraz przetwarzanie danych z czujników otoczenia
3. Wysyłania przetworzonych danych na serwer

Do realizacji tych zadań często wykorzystywałem istniejące rozwiązania, jak w przypadku obsługi czujników pobierających dane atmosferyczne.

4.1 Sterowanie silnikami DC

Sterownik DRV8833 przyjmuje 4 wejścia PWM do sterowania silnikami prądu stałego. Każde z przypadających na jeden silnik tych wejść jest oznaczone dodatkowo numerem. W celu poprawnej obsługi tych silników przez mikrokontroler zdefiniowałem 4 zmienne przypisujące wejścia sterownika DC do pinów wyjściowych przy ESP-32. Pokazano to w kodzie źródłowym nr 1.

```
const int IA1 = 13;
const int IA2 = 12;
const int IB2 = 14;
const int IB1 = 27;
```

Kod źródłowy 1 - Przypisanie wejść sterownika do pinów mikrokontrolera) , żródło : opracowanie własne[16]

W następnej kolejności zaimplementowano w programie funkcje, odpowiadające za ustawianie sygnału PWM na 4 pinach. Dzięki wcześniej zaimportowanej bibliotece „analogWrite.h" przegotowano dwie funkcje, które pozwalają na jazdę do przodu oraz do tyłu z ustalonym skrętem oraz prędkością ustawianą przez sterowanie PWM. Przykładowe użycie pokazano w kodzie źródłowym nr 2.

```
#include "analogWrite.h"
...
analogWrite(IA1,0, 100, 10, 0);
```

Kod źródłowy 2 - Przykład metody analogWrite z argumentami (nrPinu, wartość_sygnału, częstotliwość_zmiany_stanu, rozmiar_zmiennej_wartości_w_bitach, początkowa_faza);dla powyższego przypadku można nadać wartość sygnału od 0 do 1024; źródło : opracowanie własne[16]

```
//Jazda do przodu z ustalonym skrętem  na podstawiem turnParameter
void ForwardWithTurning(int baseSpeed,float turnParameter){
  analogWrite(IA1,baseSpeed *(0.5 - turnParameter), 100, 10, 0);
  analogWrite(IA2,0, 100, 10, 0);
  analogWrite(IB1,baseSpeed *(0.5 + turnParameter), 100, 10, 0);
  analogWrite(IB2,0, 100, 10, 0);
}
//Jazda do tyłu z ustalonym skrętem na podstawiem turnParameter
void BackwardWithTurning(int baseSpeed,float turnParameter){
  analogWrite(IA1,0, 100, 10, 0);
  analogWrite(IA2,baseSpeed *(0.5 + turnParameter), 100, 10, 0);
  analogWrite(IB1,0, 100, 10, 0);
  analogWrite(IB2,baseSpeed *(0.5 - turnParameter), 100, 10, 0);
}
```

Kod źródłowy 1 - Kod wyznaczający stan PWM na 4 pinach w celu ustawienia jazdy do tyłu lub do przodu. , źródło : opracowanie własne[16]

W kodzie źródłowym nr 3 przedstawiono funkcje przyjmujące dwa argumenty. Pierwszy argument *baseSpeed* służy do ustalenia maksymalnej wartości, jaka ma zostać nadana danemu wyjściu PWM. Dzięki temu argumentowi można ustawić maksymalną prędkość, z jaką może się poruszać robot. Drugi argument *turnParameter* służy do ustalenia stopnia skrętu, a jego wartość powinna się znajdować w zakresie <-0,5; 0,5> . Aby robot mógł wykonać skręt w daną stronę, to należy zadbać o to, by koło znajdujące się po stronie robota, w którą chcemy skręcić, kręciło się z mniejszą prędkością, niż koło znajdujące się po przeciwnej stronie robota. Dlatego argument jest

wykorzystywany od obliczenia prędkości obrotu każdego z kół. Dla wartości bliskiej 0,0 przy jeździe do przodu koła są napędzane z jednakową mocą. Z kolei w przypadku wartości skrajnych dla zakresu <-0,5;0,5> różnica wartości stanów jest dużo większa.

Metody odpowiadające za jazdę do przodu i do tyłu są zaimplementowane w trybie fast Decay (zgodnie z informacjami zawartymi w Tabelce nr 1). Dla jazdy do przodu napięcia przy wejściach oznaczonych numerem 1 są ustalane przez sygnał PWM, przy drugim wejściu ustawione na stan niski. Dla jazdy do tyłu sytuacja jest odwrotna. Sygnał PWM jest nadawany na wejściu nr 2, a wejście nr 1 jest ustawione na stan niski.

Dzięki tak zaprogramowanemu modelowi jazdy, robot może poruszać się do przodu i do tyłu wraz z możliwością skrętu.

4.2 Odczyt danych z czujników

Do pobierania danych z czujników w większości wypadków wykorzystywano istniejące rozwiązania dla platformy Arduino. Posiadają one już wbudowane rozwiązania co do aktywowania czujnika na czas pomiaru, pobierania danych z niego oraz przetworzenia pobranej informacji do postaci bardziej zrozumiałej. Dane z czujników są sprowadzane do postaci zmiennej float, które to następnie są wysyłane protokołem http na serwer agregujący pobrane informacje.

4.2.1 Odczyt danych z czujnika temperatury i wilgotności DHT-11

Dla czujników z rodziny DHT jest dedykowana biblioteka[17] w bazie rozwiązań Arduino. Jej zastosowanie polega na inicjalizacji struktury *DHT* obsługującej czujnik.

```
#define DHTTYPE DHT11
const int tempInput = 18;
DHT dht(tempInput, DHTTYPE);

void setup() {
  ...
  dht.begin();
  ...
}
```

Kod źródłowy 2 - Inicjalizacja struktury DHT na pinie podpiętym do czujnika, źródło : opracowanie własne[16]

W celu inicjalizacji należy ustawić nr pinu, który jest podpięty do wyjścia czujnika, a także model czujnika DHT (w naszym przypadku jest to DHT-11). Implementację pokazano w kodzie źródłowym nr 4.

Pobieranie danych o temperaturze i wilgotności (pokazane w kodzie źródłowym nr 5) sprowadza się do użycia bezargumentowych metod struktury *readHumidity* i *readTemperature*.

```
char tempMess[64];

char* getTempMess(){
  float h = dht.readHumidity();
  float t = dht.readTemperature();
  sprintf(tempMess, "Humidity: %f , °C %f", h,t);
  return tempMess;
}
```

Kod źródłowy 3 - Metoda zwracająca pobrane dane o wilgotności i temperaturze (w stopniach Celcjusza) , źródło : opracowanie własne[16]

4.2.2 Odczyt danych z czujnika odległości HC-SR04

Do obsługi czujnika odległości stworzono algorytm bez udziału biblioteki zewnętrznej. W robocie zostały umieszczone 3 czujniki tego typu. W celu obsługi ich wszystkich jednocześnie najpierw należało ustawić stany pinów do nich przypisane: OUTPUT dla wyjścia Trigger oraz INPUT dla wyjścia Echo. Pokazano to w kodzie źródłowym nr 6.

```
const int distanceEcho1 = 19;
const int distanceTrig1 = 21;
const int distanceEcho2 = 25;
const int distanceTrig2 = 26;
const int distanceEcho3 = 32;
const int distanceTrig3 = 33;

void setup() {
  ...
  pinMode(distanceTrig1,OUTPUT);
  pinMode(distanceEcho1,INPUT);
  pinMode(distanceTrig2,OUTPUT);
  pinMode(distanceEcho2,INPUT);
  pinMode(distanceTrig3,OUTPUT);
  pinMode(distanceEcho3,INPUT);
  ...
}
```

Kod źródłowy 4 - Ustawienie stanu pinów dla czujników odległości, źródło : opracowanie własne[16]

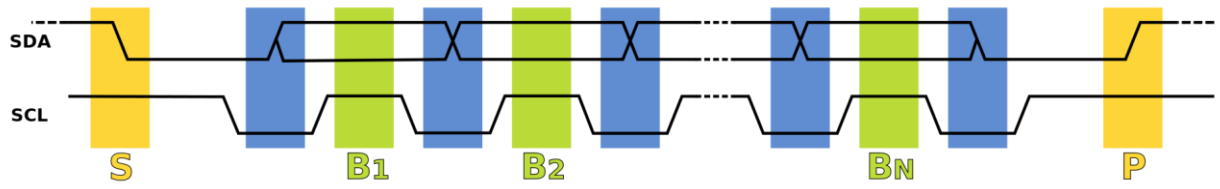
Czujnik HC-SR04 za pomocą sygnału ultradźwiękowego mierzy odległość pomiędzy czujnikiem a pierwszą przeszkodą. Sygnał dźwiękowy jest generowany w momencie, kiedy wejście czujnika Trig zostanie ustawione na stan wysoki. Następnie czujnik nadaje stan wysoki na wyjściu Echo do momentu zarejestrowania wcześniej wysłanego sygnału. Znając prędkość dźwięku oraz to, że sygnał musiał pokonać dwukrotność drogi pomiędzy czujnikiem i przeszkodą, jesteśmy w stanie wyznaczyć odległość w centymetrach, do obliczenia którego wykorzystano równanie nr 2. Implementację pokazano w kodzie źródłowym nr 7.

```
// distanceEcho oraz distanceTrig to numery pinów przypisane
do danego czujnika
float getDistance(int distanceEcho, int distanceTrig){
  // aktywowanie czujnika ultradźwiękowego
  digitalWrite(distanceTrig, LOW);
  delayMicroseconds(5);
  digitalWrite(distanceTrig, HIGH);
  delayMicroseconds(15);
  digitalWrite(distanceTrig, LOW);
  float resultIn = 0;
  //pobranie czasu, przez jaki utrzymuje sie stan wysoki
  resultIn = pulseIn(distanceEcho, HIGH);
  // podzielenie przez 58 da wartość w cm
  resultIn /= 58;
  return resultIn;
}
```

Kod źródłowy 5 - Metoda getDistance zwracająca odległość od przeszkody w centymetrach, źródło : opracowanie własne[16]

4.2.3 Odczyt danych z czujnika ciśnienia LPS331AP

Do czujnika LPS331AP wykorzystano bibliotekę LPS[18]. Pozwala ona na połączenie się z czujnikiem za pomocą interfejsu I2C oraz na pobranie danych z czujnika. Sama magistrala I2C opiera się na dwukierunkowym połączeniu cyfrowym [19] . Wyjście oznaczone jako SDA (ang. *Serial Data Line*) służy do przekazywania danych, a wyjście SCL (ang. *Serial Clock Line*) służy jako zegar połączenia (zaprezentowany na rysunku nr 13).



Rys. 13 - Przebieg czasowy sygnałów I2C, źródło: opis zagadnienia I2C na Wikipedii [20]

Aby mikrokontroler ESP-32 mógł odbierać dane przy pomocy interfejsu I2C, wpierw należy skonfigurować połączenie po stronie kontrolera. W tym celu wykorzystano bibliotekę „Wire.h", która pozwala na konfigurację dostępnych interfejsów (implementację zaprezentowano w kodzie źródłowym nr 8). Dodatkowo biblioteka LPS umożliwia wykorzystanie struktury do obsługi czujnika.

```
const int SDA0_Pin = 22;
const int SCL0_Pin = 23;
LPS ps;

void setup() {
  ...
  Wire.begin(SDA0_Pin, SCL0_Pin);

  if(!ps.init())
  {
    Serial.println("Failed to autodetect pressure sensor!");
  }
  ps.enableDefault();
  ...
}
```

Kod źródłowy 6 - ustawienie czujnika LPS331AP, źródło : opracowanie własne[16]

Pobieranie informacji odbywa się za pomocą metody struktury LPS. Informację o aktualnym stanie ciśnienia można zwrócić w postaci milibarów (która wprost przekłada się do postaci HPa) lub w postaci inHg (ang. *Inch of mercury*). Implementacja w kodzie źródłowym nr 9.

```
char pressureMess[64];
char* getPressureMess(){
  float pressure = ps.readPressureMillibars();
  // czujnik pozwala też na ustalenie wysokości w metrach n.p.m.
  float altitude = ps.pressureToAltitudeMeters(pressure);
  sprintf(pressureMess, "p:%f,inhPa\ta:%fm\t", pressure, altitude);
  return pressureMess;
}
```

Kod źródłowy 7 - funkcja zwracająca dane o ciśnieniu i na jej podstawie wyznaczonej wysokości nad poziomem morza, źródło : opracowanie własne[16]

4.2.4 Odczyt danych z czujnika niebezpiecznych gazów MQ-9

Do obsługi tego czujnika wykorzystano bibliotekę MQUnifiedsensor[21]. Pozwala ona na obsługę wszystkich czujników z rodziny MQ. W zestawie tego robota umieściłem MQ-9, który pozwala na badanie stężenia 3 rodzajów gazów.

Dane z czujnika można pobrać w czytelnej formie dzięki strukturze *MQUnifiedsensor* z biblioteki o tej samej nazwie. Struktura ta wymaga podania kilku argumentów: typu mikrokontrolera, pinu przypisanego do czujnika, rodzaju czujnika MQ, napięcia idącego do czujnika, ilość bitów przy przetwarzaniu danych analogowych oraz ustalenie domyślnej wartości stosunku RS/ R0.

Pobieranie danych o stężeniu gazu wymaga wykonania kilku metod ze struktury *MQUnifiedsensor*. Metoda *update* pobiera aktualny stan rezystywności z czujnika, który będzie wykorzystywany w dalszych obliczeniach. Do wyliczenia koncentracji danego gazu w powietrzu należy też wprowadzić dane do równania.

Na początku należy skonfigurować odbiór czujnika po stronie mikrokontrolera. Wymaga to przede wszystkim nie tylko ustawienia metod obliczających stężenie, ale także ustalenie rezystywności domyślnej dla czystego powietrza. W tym celu mikrokontroler powinien przy uruchamianiu wyprowadzić wartość R0 na podstawie średniej kilku pierwszych pomiarów. Następnie można przystąpić do pobierania danych z czujnika. Konfiguracja pokazana w kodzie źródłowym nr 10.

$$Stężenie\ gazu\ [ppm] = (RS/R0) * A + B$$

Równanie nr 3 – wzór na wyznaczenie stężenia gazu w powietrzu na podstawie ilorazu rezystywności odczytanej oraz rezystywności bazowej.

```
//Dane początkowe dla mikrokontrolera ESP-32
/************************Hardware Related
Macros************************************/
#define         Board                   ("ESP-32")
#define         Pin                     (34)  //Analog input 4 of your
arduino
/***********************Software Related
Macros************************************/
#define         Type                    ("MQ-9") //MQ9
#define         Voltage_Resolution      (3.3)
#define         ADC_Bit_Resolution      (12) // For arduino UNO/MEGA/NANO
#define         RatioMQ9CleanAir        (9.6) //RS / R0 = 60 ppm
MQUnifiedsensor MQ9(Board, Voltage_Resolution, ADC_Bit_Resolution, Pin,
Type);


void setup() {
  ...
  MQ9.setRegressionMethod(1); //_PPM =  a*ratio^b
  MQ9.init();

  float calcR0 = 0;
  for(int i = 1; i<=10; i ++)
  {
    MQ9.update();
    calcR0 += MQ9.calibrate(RatioMQ9CleanAir);
  }
  MQ9.setR0(calcR0/10);

  ...
}
```

Kod źródłowy 8 - konfiguracja obsługi czujnika MQ-9, źródło : opracowanie własne[16]

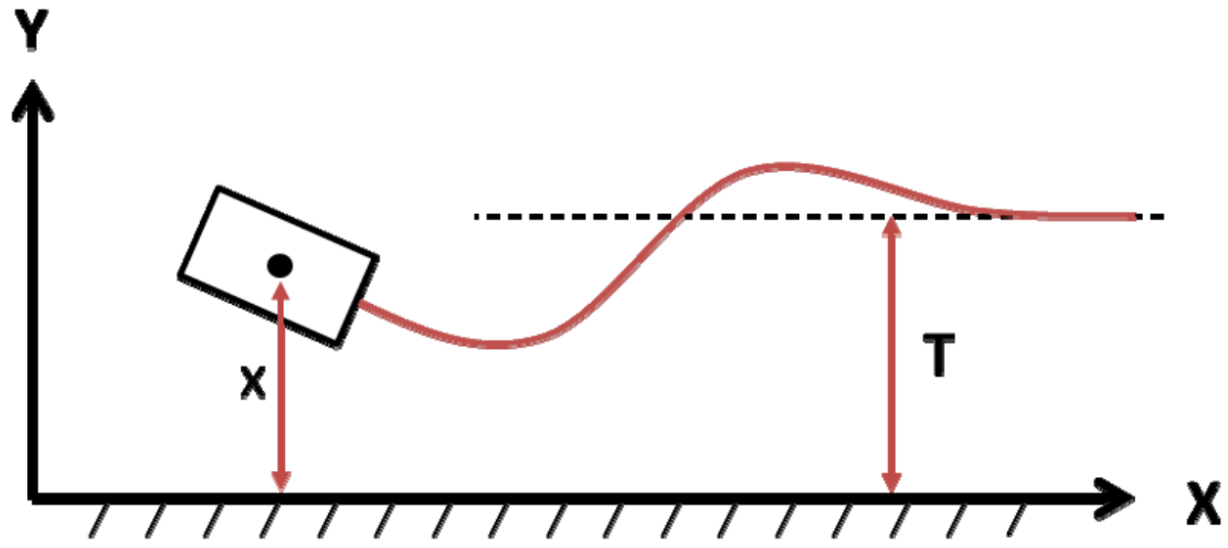
Aby otrzymać stężenie gazu należy na podstawie wykresu nr 1 dobrać współczynniki liniowe A i B. Dzięki podanemu przykładowi w repozytorium biblioteki[22] wartości są już wyznaczone. Implementacja w kodzie źródłowym nr 11.

```
float getGasSensorData(float a, float b){
  MQ9.update(); // Update data, the arduino will be read the voltage on the
analog pin
  MQ9.setA(a);MQ9.setB(b);
  return MQ9.readSensor(); // Sensor will read PPM concentration using the
model and a and b values setted before or in the setup
}
...
void loop() {
  ...
  float co = getGasSensorData(599.65,-2.244); // tlenek węgla
  float lpg = getGasSensorData(1000.5,-2.186); // gaz petrochemiczny, butan
  float ch4 = getGasSensorData(4269.6,-2.648); // Metan
  ...
}
```

Kod źródłowy 9 - pobranie danych z czujnika MQ-9, źródło : opracowanie własne[16]

4.3 Oprogramowanie funkcjonalności wall-follower

Jednym z istotnych założeń tego projektu jest zaimplementowanie sposobu autonomicznego poruszania się w przestrzeni. Robot powinien, przynajmniej w zamkniętym pomieszczeniu, móc poruszać się bez udziału i kontroli człowieka. Dlatego w tym celu dobrałem jedno z prostszych rozwiązań istniejących na rynku, czyli model poruszania się typu wall-follower[23].

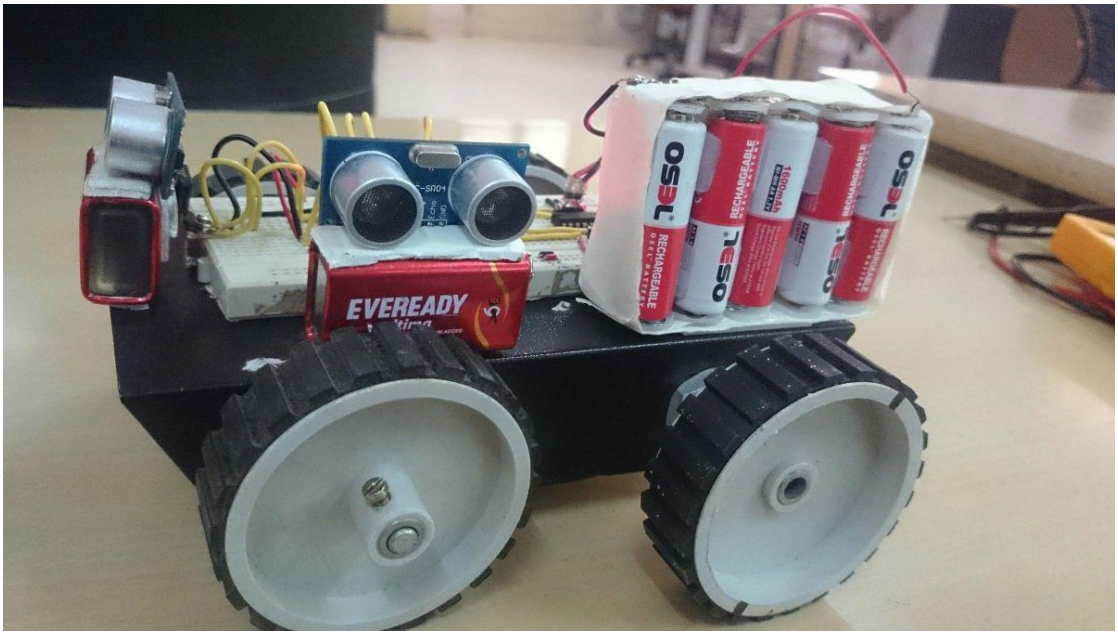


Rys. 14 - Zobrazowanie zachowania robota w metodzie wall-follower robot stara się utrzymać odległość od ściany o wartości T, źródło: praca na temat metody wall-follower [23]

Metoda (zilustrowana na rysunku nr 14) ta polega na poruszaniu się w zadanej odległości od ściany (z lewej lub prawej strony patrząc w kierunku ruchu robota) i próbie utrzymania tej odległości przy pomocy korekt (małych skrętów). Przy napotkaniu przeszkody z przodu robot powinien skręcić w stronę przeciwną, do której utrzymuje stały dystans. Robot powinien analizować odległość przy pomocy minimum dwóch czujników odległości i następnie dokonywać decyzji o korekcie skrętu, czy też samym skręcie w celu uniknięcia przeszkody.

4.3.1 Istniejące przykłady robotów typu wall-follower

Prototypy tego rodzaju robotów są bardzo popularne, głównie ze względu na prostotę implementacji oraz niedrogich komponentów wymaganych do stworzenia robota. Jednym z istniejących przykładów jest prototyp stworzony przez Hai Prasaath'a[24] (pokazany na rysunku nr 15). Oparty na Arduino przykład posiada dwa czujniki z przodu oraz z lewej strony oraz napęd silników IC. Spełnia on wszystkie założenia poprawnie stworzonego robota wall-follower. Stara się utrzymywać stałą odległość od lewej ściany.

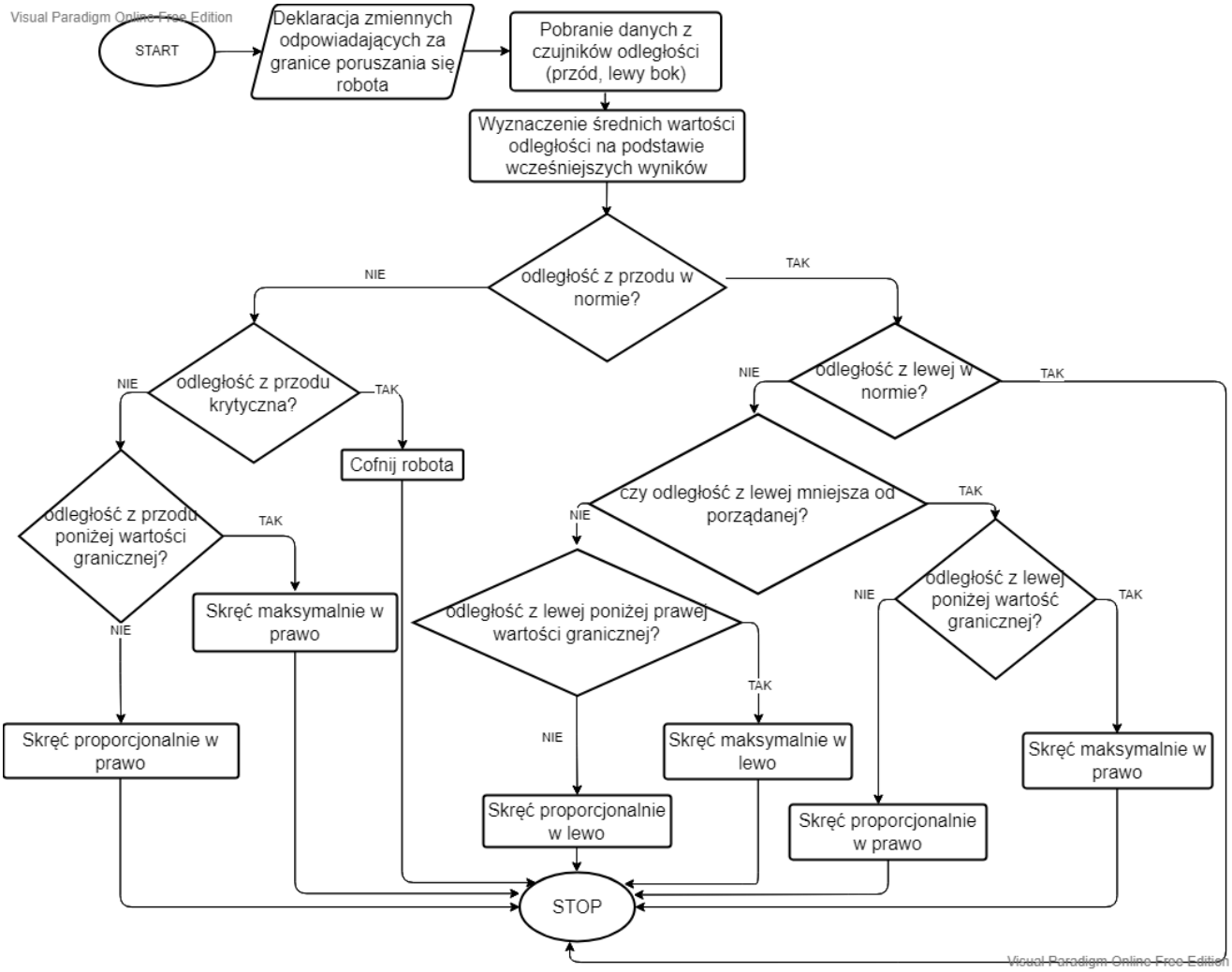


Rys. 15 - Prototyp robota typu wall-follower, źródło: opis prototypu [24]

Elektronika tego robota oparta jest o mikrokontroler Arduino Pro Mini. Istnieją dwa rodzaje zasilania w tym robocie, tego względu, że silniki wymagają napięcia 12 V, zaś same czujniki odległości tylko 5 V. W przeciwieństwie do tego przykładu, w tym robocie umieszczono 3 czujniki odległości (1 z przodu, 2 po bokach robota). Aktualnie oprogramowanie nie korzysta z 3 czujnika, jednakże biorąc pod uwagę możliwość rozwoju modelu jazdy, został on pozostawiony w robocie.

4.3.1 Własna implementacja

Wykorzystując dane z czujników odległości, zaimplementowano w tym robocie autorski algorytm wall-followingu. W tym celu najpierw dane z czujników są pobierane , a następnie przetwarzane i wprowadzane do algorytmu. Algorytm ten wyprowadza przede wszystkim wartość *turningParameter*, która jest wprowadzana do funkcji opisanych w rozdziale o sterowani silnikami DC. Posiada ona dane z zakresu < -0,5;0,5>. Wewnątrz programu zdefiniowane są zmienne *leftStandardDistance*, *leftBorder*, *frontStandardDistance*, *frontMax* oraz *rightBorder*. Decydują one w jakich granicach odległości z przodu oraz z lewego boku ma poruszać się robot. Dokładny sposób obliczania zmiennej wyjściowej pokazuje schemat blokowy na rysunku nr 16 oraz kod źródłowy nr 12, 13 i 14.



Rys. 16 - Schemat blokowy algorytmu wyznaczającego turningParameter, źródło: opracowanie własne na podstawie Visual Paradigm Online [29]

```
//wyznaczanie średniej z zakresu
float medium(float* arr, int count){
  float sum = 0.0;
  for(int i=0;i< count;i++){
    sum += arr[i];
  }
  return sum / count;
}
//dodawanie nowej informacji do tablicy
void writeRecord(float* arr, int count, float value){
  for(int i=0;i< count-1;i++){
    arr[i] = arr[i+1];
  }
  arr[count-1] = value;
}

float turningParameter = 0.0;

//zdefiniowane granice zachowań w algorytmie wall-following
float leftBorder = 5.0;
float leftStandardDistance = 20.0;
float rightBorder = 40.0;
float frontMax = 10.0;
float frontBorder = 20.0;
float frontStandardDistance = 30.0;

//zapis ostatnio zmierzonych odległości
float lastRecordsFront[] = {0.0, 0.0, 0.0, 0.0, 0.0 };
float lastRecordsLeft[] = {0.0, 0.0, 0.0, 0.0, 0.0 };

void loop() {
  float left = getDistance(distanceEcho1,distanceTrig1);
  float front = getDistance(distanceEcho2,distanceTrig2);
  float right = getDistance(distanceEcho3,distanceTrig3);
  ...
  writeRecord(lastRecordsFront,5,front);
  writeRecord(lastRecordsLeft,5,left);
  front = medium(lastRecordsFront,5);
  left = medium(lastRecordsLeft,5);
  //wall-following
}
```

Kod źródłowy 10 - algorytm przetwarzania wartości z czujników odległości, źródło : opracowanie własne[16]

```
bool back = false;
  if(front > frontStandardDistance){
    //skręt w prawo
    if(left < leftStandardDistance){
      float diff = leftStandardDistance - left;
      if(diff > leftStandardDistance - leftBorder){
        turningParameter = 0.5;
      }
      else{
        turningParameter = 0.5 * (leftStandardDistance -
left)/(leftStandardDistance - leftBorder);
      }
    }
    //skręt w lewo
    else if(left > leftStandardDistance){
      float diff = left - leftStandardDistance;
      if(left > rightBorder){
        turningParameter = -0.5;
      }
      else{
        turningParameter = -0.5 * (rightBorder - leftStandardDistance +
(diff - leftStandardDistance))/(rightBorder - leftStandardDistance);
      }
    }
  }
  //cofanie się z zachowaniem skrętu
  else{
    float diff =  frontStandardDistance - front;
      if(front < frontMax){
        back = true;
      }
      else if(front < frontBorder){
        turningParameter = 0.5;
      }
      else{
        turningParameter = 0.5 * (frontStandardDistance -
front)/(frontStandardDistance - frontBorder);
      }
  }
```

Kod źródłowy 11 - algorytm wall-following, wyznaczanie wartości turningParameter, źródło : opracowanie własne[16]

```
//cofanie musi trwać przynajmniej sekundę, by robot wydostał się z
zaklinowanego miejsca
  if(back){
    BackwardWithTurning(1024,turningParameter);
    delay(1000);
    back = false;
  }
  ForwardWithTurning(1024,turningParameter);
```

Kod źródłowy 12 - ruszanie silnikami DC na podstawie wyliczonych parametrów, źródło : opracowanie własne[16]

4.3.1 Algorytm PID

W celu wyznaczenia wartości turningParameter zaimplementowano dwie z 3 części algorytmu PID, czyli algorytm proporcjonalno-całkująco-różniczkujący (ang. proportional–integral–derivative) [25] .

Pierwsza część członu odnosi się do stopniowania proporcjonalnego danej wartości od innej wartości wejściowej. Za przykład może posłużyć wzór wyznaczania parametru skrętu na podstawie odległości od lewej wartości granicznej (patrz Kod źródłowy 13). W algorytmie rolę danej wyjściowej pełni turningParameter, a danej początkowej odległość od wartości granicznej Im większa odległość od lewej wartości granicznej (odległości, po której skręt w prawo powinien być jak najostrzejszy) tym mniejszy stopień skrętu w prawą stronę. Dzięki tej technice udaje się zasymulować płynny skręt podczas korygowania odległości i program nie musi operować tylko na wartościach skrajnych -0,5 lub 0,5.

Druga część członu odnosi się do analizy całkującej. Do algorytmu Wall-Followera nie przesyła się aktualnie pobranej danej odległości, tylko jego uśrednioną wartość z ostatnich 10 pomiarów. Ta technika pozwala zapobiegać jednokrotnym skrajnym wahnięciom wartości, które mogą wynikać np. z nagłych usterek, czy innych problemów. Dzięki temu wartości wchodzące do algorytmu są dużo mniej podatne na wahnięcia i zakłócenia.

4.4 Komunikacja z serwerem zewnętrznym

Kolejnym elementem oprogramowania, jaki zaimplementowano w tym prototypie, jest komunikacja z zewnętrznym serwerem API przyjmującym zapytania http. Mikrokontroler ESP-32 posiada w sobie wbudowany moduł WiFi w paśmie 2,4 GHz [2]. Jest to jeden z bardzo tanich mikrokontrolerów posiadających obsługę takiej komunikacji.

Komunikacja jest obsłużona na płytce dzięki dwóm bibliotekom „WiFi.h" oraz „"HTTPClient.h", które są już zaimplementowane w bibliotece obsługującej płytkę[14].

4.4.1 Moduł WiFi

Robot powinien zacząć komunikację od połączenia się z aktualnie dostępną siecią WiFi. Do tego celu wykorzystano funkcje już istniejące w bibliotece (pokazano w kodzie źródłowym nr 15).

```
char* ssid = "ssid";
char* password = "password";

void setup() {
  ...
  WiFi.begin(ssid, password);
  ...
}
```

Kod źródłowy 13 - połączenie do sieci WiFi

Część poradników proponuje na początku programu odczekanie i upewnienie się, czy połączenie zostało dokonane. Ze względu jednak na to, że robot musiałby czekać z kilka sekund w bezruchu na połączenie z siecią bezprzewodową, nie wprowadzono tego rozwiązania. Dopiero przy wysyłaniu wiadomości programu powinien sprawdzić, czy udało się połączyć z siecią WiFi (warunek pokazany w kodzie źródłowym nr 16).

```
if ((WiFi.status() == WL_CONNECTED))
{
  ...
}
```

Kod źródłowy 14 - sprawdzenie, czy udało się nawiązać połączenie z siecią

W celu uniknięcia problemów z poszukiwaniem odpowiedniej sieci przyjęto założenie, że w wersji prototypowej dane logowania do autoryzowanej sieci WiFi będą zapisane w kodzie programu. Mimo wielu wad tego rozwiązania, pozwala ono w najszybszy sposób przetestować oprogramowanie.

4.4.2 Wysyłanie danych na serwer

Wysyłanie danych za pomocą protokołu http polega na złożeniu zapytania http oraz jego wykonania. Do wysyłania danych na serwer na konkretny adres URL wykorzystuję metodę POST, zgodnie z zasadą REST API. Do złożenia zapytania i jego wykonania zaimplementowano osobną funkcję (w kodzie źródłowym nr 17).

```
String prodUrl = "http://93.180.174.50:180/api/devices/122/add-values-by-
property-id";

char POSTObject[64];
void WiFiPost(String property,float val){
  if ((WiFi.status() == WL_CONNECTED))
  {
    //Utworzenie struktury http
    HTTPClient http;
    //zapisanie danych w postaci JSON, przy pomocy metod obsługi ciągu
znaków
    sprintf(POSTObject, "{ \"propertyId\":\"%s\" , \"val\":\"%f\" }",
property, val);
    http.begin(wifiUrl);
    //oznaczenie, zę dane są w postaci JSON
    http.addHeader("Content-Type","application/json");
    int httpCode = http.POST(POSTObject);
    http.end();
    if( httpCode > 200){
      //wykonanie w przypadku poprawnej odpowiedzi, dla potrzeb tego robota
nie było potrzeby upewniania się, czy dane zapisały się poprawnie
    }
  }
}
```

Kod źródłowy 15 - funkcja wysyłania zapytań POST

W kodzie źródłowym nr 18 pokazano przykład wysłanego pliku JSON, który ma być wysyłany z zapytaniem http.

```
{
    "propertyId":"128",
    "val":"22"
}
```

Kod źródłowy 16 - przykładowy obiekt JSON przesyłany podczas wysyłania danych na serwer

Robot nie ma potrzeby, by weryfikował poprawność przesłania danych, z tego powodu, że nie ma możliwości na dłuższą metę przechowywania wszystkich danych brakujących na serwerze. Dodatkowo w przypadku tego serwera, nie można przestawiać kolejności agregacji danych, przez co przechowywanie tym bardziej jest bezcelowe. Częściowo więc przesyłanie danych przypomina przesyłanie zapytań protokołem UDP.

4.5 Sterowanie ręczne przy pomocy interfejsu Bluetooth

W pracy, poza autonomicznym poruszaniem się robota, została też zaimplementowana możliwość sterowania ręcznego. Dzięki połączeniu przez technologię Bluetooth (pokazano to w kodzie źródłowym nr 19) oraz aplikacji umożliwiającej przesyłać komunikaty za jej pomocą, użytkownik może wydawać bezpośrednie polecenia danemu robotowi. Podczas przyjmowania komunikatu robot wstrzymuje wykonywanie zachowania autonomicznego. Zamiast tego przetwarza wiadomość przekazaną przez Bluetooth i wykonuje operację przypisaną do danego komunikatu.

```
#include "BluetoothSerial.h"
//Połączenie bloetooth do sterowania ręcznego
BluetoothSerial SerialBT;

void setup() {
  ...
  SerialBT.begin("ESP32-PI-Robot");
  ...
}
```

Kod źródłowy 17 - nadawanie sygnału Bluetooth

W kodzie źródłowym nr 21 pokazano zaimplementowane cztery funkcje realizujące proste ruchy: ruch do przodu, ruch do tyłu, skręt w prawo oraz w lewo. Każde polecenie Bluetooth (obsłużone przez kod źródłowy nr 20) wykonuje się przez minimum sekundę i trwa dopóki komunikat Bluetooth nie przestanie nadawać. Tego rodzaju rozwiązanie pozwala uzależnić stan sterowania ręcznego od tego, czy użytkownik rzeczywiście znajduje się w pobliżu robota i świadomie kontroluje jego ruch.

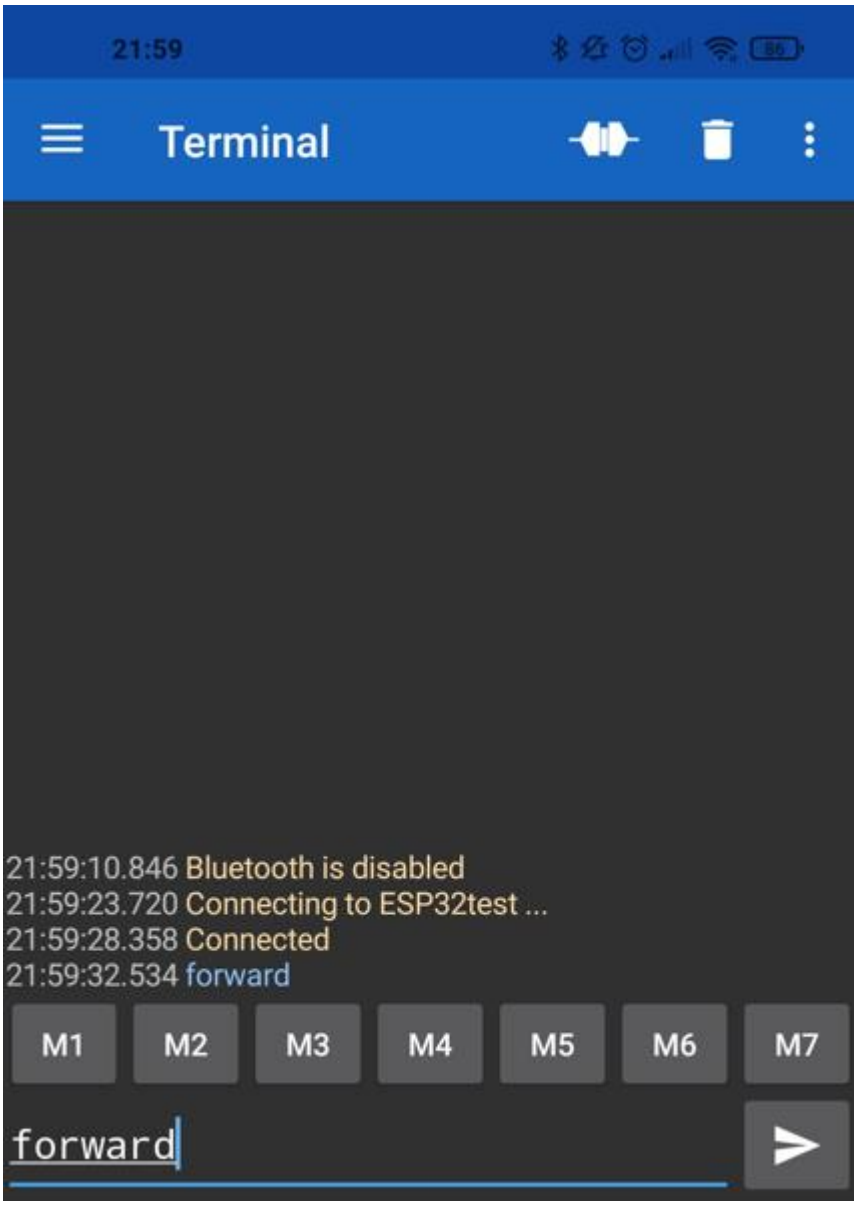
```
if(SerialBT.available()){
    bluetoothCommand = SerialBT.readString();
    //czy komunikat jest identyczny dla kodu operacji
    if(strcmp(bluetoothCommand.c_str(), "forward") == 0){
      Forward(1024); delay(1000);
    } else if(strcmp(bluetoothCommand.c_str(), "backward") == 0){
      Backward(1024); delay(1000);
    } else if(strcmp(bluetoothCommand.c_str(), "turn-left") == 0){
      TurnLeft(1024); delay(1000);
    } else if(strcmp(bluetoothCommand.c_str(), "turn-right") == 0){
      TurnRight(1024);delay(1000);
    }
  }
```

Kod źródłowy 18 – sterowanie robotem na podstawie komend

```
  void Forward(int speed){
    analogWrite(IA1,speed, 100, 10, 0);
    analogWrite(IA2,0, 100, 10, 0);
    analogWrite(IB1,speed, 100, 10, 0);
    analogWrite(IB2,0, 100, 10, 0);
  }
  void Backward(int speed){
    analogWrite(IA1,0, 100, 10, 0);
    analogWrite(IA2,speed, 100, 10, 0);
    analogWrite(IB1,0, 100, 10, 0);
    analogWrite(IB2,speed, 100, 10, 0);
  }
  void TurnLeft(int speed){
    analogWrite(IA1,speed, 100, 10, 0);
    analogWrite(IA2,0, 100, 10, 0);
    analogWrite(IB1,0, 100, 10, 0);
    analogWrite(IB2,0, 100, 10, 0);
  }
  void TurnRight(int speed){
    analogWrite(IA1,0, 100, 10, 0);
    analogWrite(IA2,0, 100, 10, 0);
    analogWrite(IB1,0, 100, 10, 0);
    analogWrite(IB2,speed, 100, 10, 0);
  }
```

Kod źródłowy 19 – podstawowe funkcje ruchu nadające sygnał PWM według specyfikacji w Tabeli nr 1

Do testowania sterowania ręcznego została zastosowana aplikacja Serial Bluetooth Terminal[30] (pokazana na rysunku nr 17). Po połączeniu z robotem aplikacja wysyłała komunikat z wiersza poleceń. Robot po ponad sekundzie z trybu automatycznego przełączył się na tryb ręczny i następnie wykonał zadane polecenie.



Rys. 17 - Zrzut ekranu aplikacji Serial Bluetooth Terminal, źródło: opracowanie własne

5. Zastosowania

W ramach rozdziału o tytule „Cel Pracy” opisano całościowo pomysł na projekt inżynierski. Projekt składający się z przenośnego zestawu autonomicznych robotów oraz serwera miałby mieć ogólnie określone zastosowania. Dokładniej mowa tutaj o zastosowaniach w wielu unikatowych przypadkach. Dzięki mobilności projektu można by wykonać zestaw pomiarów dowolnego rodzaju w wielu budynkach, które nie posiadają odpowiedniej architektury pomiarowej. Pomiary dowolnego rodzaju byłyby możliwe, dzięki przystosowaniu prototypu do działania modularnego, to znaczy do możliwości prostej wymiany czujników. Dzięki temu rozwiązaniu zestaw ten mógłby być dostosowywany do konkretnych potrzeb bez potrzeby tworzenia nowego zestawu.

Przy dopracowaniu modularności systemu, zestaw byłby modyfikowalny nie tylko z poziomu serwisanta, ale także z poziomu samego użytkownika. Tego rodzaju podejście pozwalało by na realizację wszelkich pomiarów atmosferycznych (i nie tylko). W zależności od potrzeb konkretnego użytkownika mógłby on zastosować ten zestaw w celu rozwiązania problemu, który:

1. Wymaga możliwości objęcia dużego obszaru zamkniętego bez potrzeby tworzenia infrastruktury (instalacji elektrycznej oraz sieciowej).
2. Wymaga mobilności, która pozwala zastosować zestaw w wielu miejscach
3. Wymaga zachowania bezpieczeństwa poprzez zachowanie odpowiedniego dystansu

Trzeba pamiętać, że klasyczne instalacje pomiarowe mogą być dużo wydajniejsze w realizacji swoich zadań niż tego rodzaju zestawy, jednakże wykorzystując atut mobilności i bezpieczeństwa można odnaleźć przypadki, gdzie tego rodzaju zestaw znajdzie zastosowanie.

Atutem tego projektu jest to, że umożliwia na wszechstronne sposoby alarmowania użytkowników o potencjalnym zagrożeniu, na przykład w wyniku wykrycia niebezpiecznego stężenia trujących gazów. W części prototypowej informuje o niebezpieczeństwie przekazuje w postaci sygnału świetlnego na robocie oraz sygnału przez interfejs Bluetooth. W planowanej części serwerowej alarmowanie uwzględnia wysłanie wiadomości e-mail z informacją o zagrożenia lub też powiadomienie przez aplikację WWW.

6. Dalsze możliwości rozwoju projektu

Prototyp, jaki został stworzony w ramach tej pracy inżynierskiej jest tylko elementem większej całości. W ramach dokładniejszych testów skorzystano z innej pracy, która pełniła rolę serwera w całościowym projekcie.

Aby projekt osiągnął swój zamierzony cel, należałoby dodać kilka istotnych elementów, takich jak nowe zestawy czujników składające się z wykrywaczy dymu oraz łatwopalnych gazów typu MQ-2 lub czujniki czystości powietrza. Wprowadzenie nowego rodzaju robota mogłoby sprawdzić, czy działanie wielu tego rodzaju robotów byłoby efektywne na przykład przy prowadzeniu pomiarów większej powierzchni.

Konstruując nowy prototyp należałoby wprowadzić inny solidniejszy napęd elektryczny oraz większe koła. Aktualny napęd zawarty w prototypie spełniają swoją role, ale głównie w małych pomieszczeniach. Pozwoliłoby to na umożliwienie poruszania się po trudniejszym terenie.

Robot powinien być też przystosowany do modularnej wymiany czujników przymocowanych do robota. Można to osiągnąć wprowadzając oprogramowanie wykrywające czujnik i dobierające sposób komunikacji, albo poprzez stworzenie pośrednich ustandaryzowanych nakładek na czujniki.

Ostatnim elementem brakującym w tym projekcie jest serwer kontrolujący roboty w terenie. Tą rolę mógłby pełnić komputer przenośny z podpiętą dalekosiężną anteną WiFi [1] . Mógłby on na bieżąco łączyć się z robotami, umożliwiać im możliwość podłączenia do internetu oraz wydawać im polecenia.

7. Podsumowanie

Celem niniejszej pracy było zrealizowanie prototypu robota mobilnego, który jest realizacja części większego pomysłu. W tej pracy skupiono się na części robotycznej projektu, a dokładniej na zbieraniu oraz wysyłaniu danych, a także na oprogramowania autonomicznego zachowania .

Robot posiada możliwość dokonywania pomiarów 6 parametrów:

- temperatury otaczającego go powietrza

- wilgotności powietrza

- ciśnienia atmosferycznego

- stężenia gazów: tlenku węgla, gazu petrochemicznego oraz metanu.

Robot pobrane dane przetwarza do postaci jednostek SI, a następnie dane wysyła na zewnętrzny serwer.

Realizacja tego prototypu wymagała stworzenia instalacji elektronicznej na tyle kompatybilnej z gabarytami tego robota, aby mógł on bez najmniejszych problemów poruszać się po przestrzeni zamkniętej i aby wszystkie czujniki miały możliwość wykonywania pomiarów.

Dodatkowo robot wymagał też implementacji autonomicznego poruszania się. Dzięki wprowadzeniu algorytmu wall-following, robot może przemieszczać się po obszarze zamkniętym i unikać przeszkód. Autonomiczność pozwala robotowi na wykonywanie pomiarów bez udziału człowieka i pełną automatyzację procesu zbierania danych w przyszłości. W razie konieczności, dzięki interfejsowi Bluetooth, użytkownik może przejąć kontrolę na robotem.

Zrealizowany projekt spełnił założenia pod kątem zbierania wielu rodzajów danych oraz samodzielności działania. Dane dotyczące parametrów atmosferycznych oraz stężeń gazów są odbierane z czujników oraz wysyłane na dedykowany serwer w przetworzonej postaci. Robot potrafi poruszać się w dwóch trybach: autonomicznych oraz ręcznym.

Podsumowując prototyp ten jest zakończonym pierwszym krokiem w realizacji wcześniej już opisanego pomysłu.

Bibliografia

[1] Strona opisującą dalekosiężną antenę WiFi - https://tienda.siliceo.es/en/wifi-panel-antenna/435-melon-n519d-wifi-adapter-usb-ac-panel-antenna.html

[2] Dokumentacja modułu ESP32-WROOM-32 - https://www.espressif.com/sites/default/files/documentation/esp32-wroom-32e_esp32-wroom-32ue_datasheet_en.pdf

[3] Opis ESP32 w sklepie Botland - https://botland.com.pl/moduly-wifi-i-bt-esp32/8893-esp32-wifi-bt-42-platforma-z-modulem-esp-wroom-32-zgodny-z-esp32-devkit-5904422337438.html

[4] Dokumentacja techniczna ESP32-Devkit - https://www.espressif.com/sites/default/files/documentation/esp32-devkitc-v4_reference_design.zip

[5] Tinkercat - https://www.tinkercad.com/

[6] Opis DRV8833 na stronie firmy Polulu - https://www.pololu.com/product/2130

[7] Dokumentacja techniczna DRV8833 - https://www.ti.com/lit/ds/symlink/drv8833.pdf

[8] Strona sprzedaży czujnika DHT-11 w sklepie Botland - https://botland.com.pl/czujniki-multifunkcyjne/1886-czujnik-temperatury-i-wilgotnosci-dht11-modul-przewody-5903351242448.html

[9] Dokumentacja czujnika DHT-11 - https://botland.com.pl/index.php?controller=attachment&id_attachment=251

[10] Strona sprzedaży czujnika odległości HC-SR04 - https://botland.com.pl/ultradzwiekowe-czujniki-odleglosci/1420-ultradzwiekowy-czujnik-odleglosci-hc-sr04-2-200cm-5903351241366.html

[11] Strona sprzedaży czujnika ciśnienia LPS331AP - https://botland.com.pl/czujniki-cisnienia/1421-lps331ap-czujnik-cisnienia-i-wysokosci-126kpa-i2c-spi-3-5v-pololu-2126-5903351249416.html

[12] Dokumentacja czujnika MQ-9 - https://www.electronicoscaldas.com/datasheet/MQ-9_Hanwei.pdf

[13] Arduino IDE - https://www.arduino.cc/en/software

[14] Biblioteka do obsługi płytki ESP-32 Devkit przez Arduino IDE - https://github.com/espressif/arduino-esp32/releases/

[15] Biblioteka do użycia metody analogWrite w mikrokontrolerze ESP-32 - https://www.arduino.cc/reference/en/libraries/esp32-analogwrite/

[16] Repozytorium mojej pracy inżynierskiej - https://github.com/MStabryla/Inzynierka/blob/master/final/final.ino

[17] Biblioteka do obsługi czujników z rodziny DHT - https://www.arduino.cc/reference/en/libraries/dht-sensor-library/

[18] Biblioteka do obsługi czujnika LPS - https://www.arduino.cc/reference/en/libraries/lps/

[19] Opis połączenia cyfrowego w książce Mielczarka W., „Szeregowe interfejsy cyfrowe” , Gliwice: Helion, 1993, ISBN 83-85701-23-0.

[20] Przebieg czasowy sygnałów I2C - https://upload.wikimedia.org/wikipedia/commons/thumb/6/64/I2C_data_transfer.svg/1920px-I2C_data_transfer.svg.png

[21] Biblioteka do obsługi czujnika Mq-9 - https://www.arduino.cc/reference/en/libraries/mqunifiedsensor/

[22] Przykład obsługi MQ-9 - https://github.com/miguel5612/MQSensorsLib/blob/master/examples/MQ-9/MQ-9.ino

[23] Praca opisująca model typu wall-follower - https://sunfest.seas.upenn.edu/wp-content/uploads/2018/07/12-bayer.pdf

[24] Opis prototypu robota typu wall-follower - https://www.engineersgarage.com/efficient-wall-following-robot-with-ultrasonic-sensor-that-works-in-both-indoor-and-outdoor-environments/

[25] – Algorytm PID - https://pl.wikipedia.org/wiki/Regulator_PID#Algorytm_regulatora

[26] Efficient Measurement Planning for Remote Gas Sensing with Mobile Robots - https://www.researchgate.net/publication/277880299_Efficient_Measurement_Planning_for_Remote_Gas_Sensing_with_Mobile_Robots

[27] “The nature of the sensors used entails that gas emissionscould go undetected due to sparse measurements and that,whenever there are changes to the environment where thenetwork is placed, a new, time-consuming deployment couldbe required” - ↑ Efficient Measurement Planning for Remote Gas Sensing with Mobile Robots ; strona 1

[28] “Moreover, their use has be-come even more appealing since the introduction of sensors which are capable of detecting gases remotely.” - ↑ Efficient Measurement Planning for Remote Gas Sensing with Mobile Robots ; strona 1

[29] Visual Paradimg Online - https://online.visual-paradigm.com/

[30] Serial Bluetooth Terminal - https://play.google.com/store/apps/details?id=de.kai_morich.serial_bluetooth_terminal&hl=en_US&gl=US

[31] Miabot Pro - https://www.robotshop.com/media/files/PDF/Miabot-Pro-User-Manual.pdf