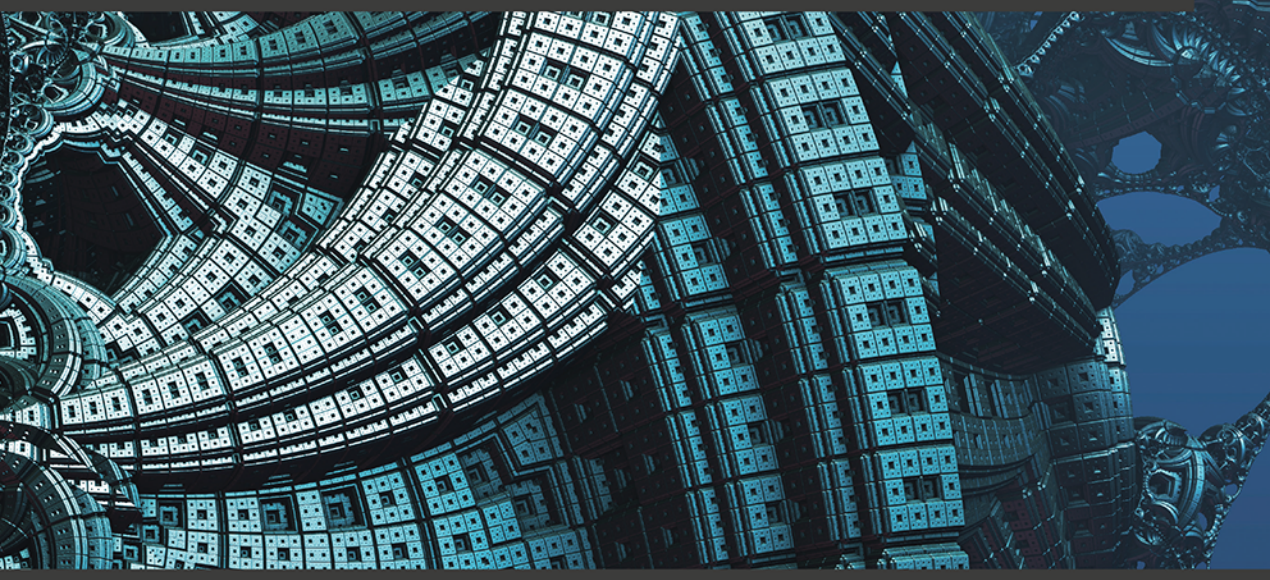
Jak zaprogramować robota

Wydanie II

Zastosowanie Raspberry Pi i Pythona
w tworzeniu autonomicznych robotów



Danny Staple

Helion

Tytuł oryginału: Learn Robotics Programming: Build and control AI-enabled autonomous robots using the Raspberry Pi and Python, 2nd Edition

Tłumaczenie: Anna Mizerska

ISBN: 978-83-283-8167-4

Helion S.A.
ul. Kościuszki 1c, 44-100 Gliwice
tel. 32 231 22 19, 32 230 98 63
e-mail: *helion@helion.pl*
WWW: *http://helion.pl* (księgarnia internetowa, katalog książek)

Pliki z przykładami omawianymi w książce można znaleźć pod adresem:
https://ftp.helion.pl/przyklady/jazar2.zip

Drogi Czytelniku!
Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres
http://helion.pl/user/opinie/jazar2
Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Printed in Poland.

Spis treści

8

ROZDZIAŁ

Programowanie czujników odległości za pomocą Pythona

W tym rozdziale przyjrzymy się czujnikom odległości i temu, jak ich używać do omijania obiektów. Unikanie przeszkód jest kluczową funkcjonalnością robotów mobilnych, gdyż wpadanie na różne rzeczy z zasady nie jest pożądanym zachowaniem. Dzięki temu zacznie wydawać się, że nasz robot zachowuje się inteligentnie.

W niniejszym rozdziale odkryjemy różne rodzaje czujników i wybierzemy odpowiedni. Następnie dodamy warstwę w naszym obiekcie robota, aby mieć dostęp do czujników. Na końcu stworzymy zachowanie, które będzie polegać na unikaniu ścian i omijaniu obiektów.

W tym rozdziale zajmiemy się następującymi tematami:

- Wybór między czujnikami optycznymi a ultradźwiękowymi.
- Podłączanie czujnika ultradźwiękowego i odczytywanie z niego danych.
- Unikanie ścian — skrypt omijania przeszkód.

Wymagania techniczne

W tym rozdziale potrzebne będą:

- robot Raspberry Pi i kod napisany w poprzednim rozdziale;
- dwa ultradźwiękowe czujniki odległości: HC-SR04P, RCWL-1601 lub Adafruit 4007, które muszą mieć napięcie wyjściowe równe 3,3 V;

- płytka stykowa;
- jednodrutowy przewód o średnicy 0,643 mm (22 AWG) lub zestaw przewodów do płytki stykowej;
- przełącznik suwakowy **SPDT (jednobiegunowy z dwoma stykami)**, który można wpiąć do płytki stykowej;
- przewody połączeniowe męsko-żeńskie, najlepiej zestaw w postaci taśmy;
- dwa wsporniki dla czujników;
- wkrętak krzyżakowy;
- miniaturowe klucze płaskie lub małe obcęgi.

Kod z tego rozdziału znajdziesz w archiwum pobranym pod adresem *https://ftp.helion.pl/przyklady/jazar2.zip*, w folderze *r08*.

Film pokazujący, jak zaprogramować czujnik odległości za pomocą Pythona, znajdziesz tutaj: *https://bit.ly/2KfCkZM*.

Wybór między czujnikami optycznymi a ultradźwiękowymi

Zanim zaczniemy używać czujników odległości, dowiedzmy się, czym są, jak działają i poznajmy kilka ich rodzajów.

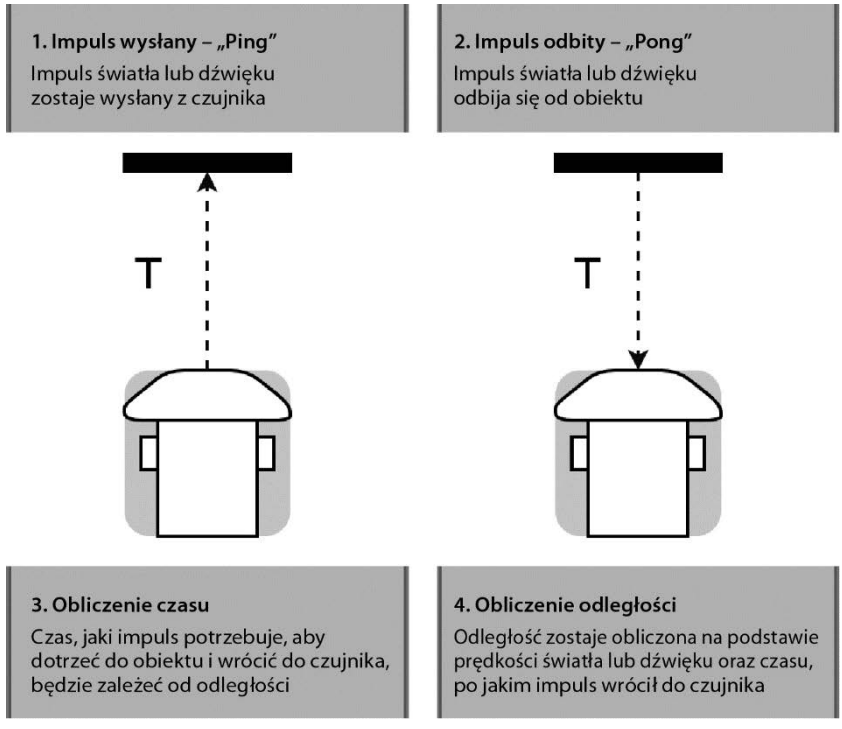
Najpowszechniej stosowanym sposobem mierzenia odległości jest ultradźwięk lub światło. Zasada działania obu tych mechanizmów polega na nadaniu impulsu, a następnie czekaniu, aż wróci po odbiciu się od przeszkody. Pomiar opiera się na czasie podróży impulsu lub kącie, co widać na rysunku 8.1.

Skupimy się na czujnikach **typu ToF** (ang. *time of flight*, czas lotu), które mierzą czas, po którym impuls wraca. Rysunek 8.1 pokazuje, jak czujniki tego typu używają czasu odbicia.

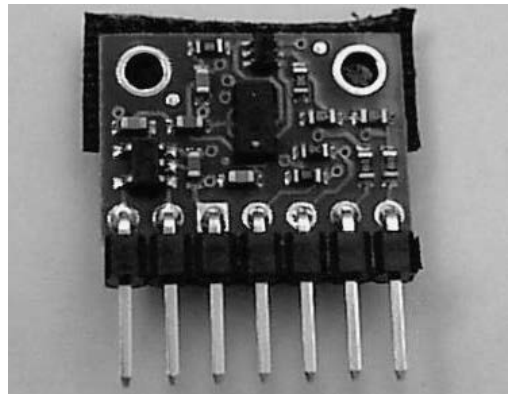
Gdy już znasz podstawową zasadę działania czujników odległości, przyjrzyj się bliżej czujnikom optycznym i ultradźwiękowym.

Optyczne czujniki odległości

Czujniki oparte na świetle, jak ten pokazany na rysunku 8.2, używają wiązki podczerwieni, która jest niewidzialna dla ludzkiego oka. Urządzenia tego typu mogą być bardzo małe, jednak mocne światło słoneczne i fluorescencyjne może sprawić, że nie będą działać prawidłowo. Niektóre obiekty słabo odbijają światło lub są przeźroczyste i nie są wykrywane przez te czujniki.



Rysunek 8.1. Mierzenie odległości na podstawie czasu



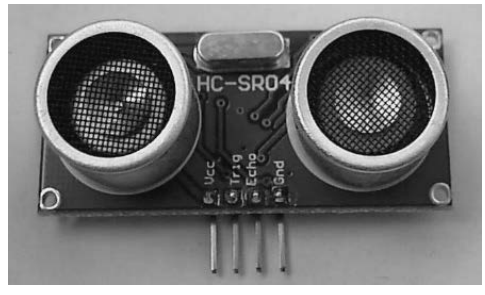
Rysunek 8.2. Moduł z czujnikiem VL530LOx

Gdy czas mierzy za pomocą wiązki światła podczerwonego więcej niż jeden czujnik, istnieje ryzyko, że wiązki i czujniki będą się zakłócać. Jednak w przeciwieństwie do czujników ultradźwiękowych nieprawidłowe pomiary są mało prawdopodobne, gdy czujniki są zainstalowane po przeciwnych stronach robota. Optyczne czujniki odległości mogą mieć większą dokładność, ale kosztem zasięgu. Niektóre z nich są drogie, ale są dostępne tańsze modele o stałym zakresie pomiaru.

Ultradźwiękowe czujniki odległości

Wiele czujników mierzących odległość za pomocą dźwięku używa ultradźwięków o częstotliwości niesłyszalnej dla ludzkiego ucha, jednak mogą one denerwować niektóre zwierzęta, w tym psy. Mikrofony telefonów komórkowych i kamery odbierają wysłane przez nie impulsy jako dźwięk klikania. Czujniki ultradźwiękowe są zazwyczaj większe od ich optycznych odpowiedników, ale tańsze, gdyż dźwięk podróżuje wolniej niż światło i łatwiej za jego pomocą zmierzyć odległość. Miękkie obiekty, na przykład tkaniny, mogą być trudne do wykrycia dla tych urządzeń.

Na rysunku 8.3 pokazano czujnik HC-SR04, popularny i niedrogi ultradźwiękowy czujnik odległości.



Rysunek 8.3. Ultradźwiękowy czujnik odległości HC-SR04

Ultradźwiękowe czujniki odległości mogą zmierzyć odległość od 2 cm do 4 m.

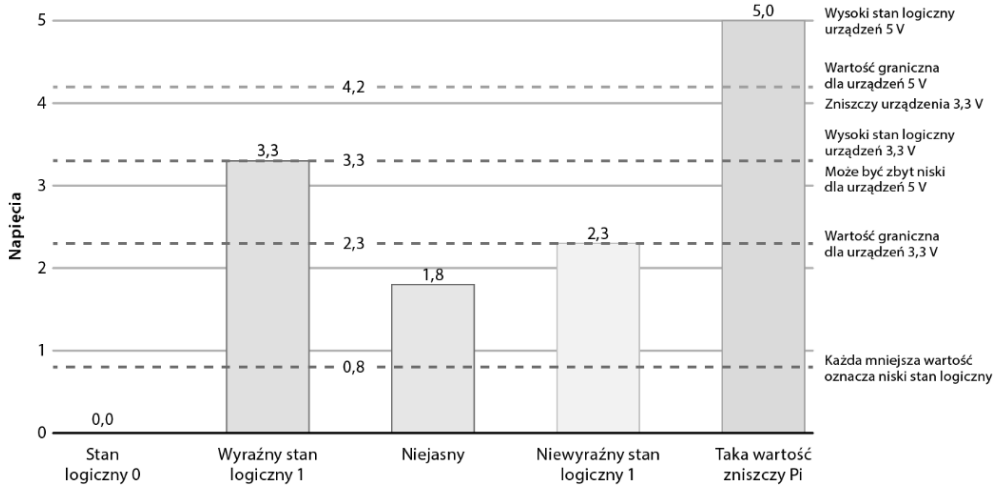
Jest wiele czujników ultradźwiękowych, w tym popularny HC-SR04, ale nie wszystkie będą pasować. Będziemy zwracać uwagę na napięcie stanów logicznych, gdyż jest to ważny czynnik przy wyborze czujnika.

Stany logiczne i przesuwanie poziomów napięcia

Piny I/O na Raspberry Pi są odpowiednie tylko dla wejść o napięciu 3,3 V. Napięcie stanów logicznych wielu dostępnych na rynku urządzeń wynosi 5 V albo dla wejść, w przypadku gdy chcemy nimi sterować, albo dla wyjść. Skupmy się na tym, co rozumiem przez stany logiczne, i na tym, dlaczego powinniśmy się trzymać napięcia określonego przez producenta urządzenia zawsze wtedy, gdy jest to możliwe.

Napięcie jest miarą energii popychającej w przepływie prądu. Różne urządzenia elektroniczne tolerują różne napięcia lub reagują na różne stany napięcia. Z jednej strony zbyt duże napięcie może zniszczyć urządzenie. Z drugiej strony zbyt niskie napięcie może skutkować brakiem odpowiedzi lub dziwnym działaniem czujników lub urządzeń wyjścia. My będziemy pracować z urządzeniami cyfrowymi, które zwracają wysokie lub niskie napięcie, aby przedstawić prawdę lub fałsz. Wartości napięcia muszą być powyżej określonego progu, aby były odczytane jako prawda, lub poniżej, aby zwrócić fałsz. Musimy znać te elektryczne właściwości urządzenia, gdyż w przeciwnym razie możemy je uszkodzić i sprawić, że nie będzie się poprawnie komunikować z innymi.

Wykres przedstawiony na rysunku 8.4 pokazuje rezultaty różnych stanów.



Rysunek 8.4. Napięcia i stany logiczne

Na rysunku 8.4 pokazano wykres. Oś *y* (po lewej stronie) to wartości napięcia z zakresu od 0 V do 5 V. Pokazuje różne warunki działania. Wzdłuż wykresu biegną cztery linie przerywane. Linia znajdująca się najbliżej osi *x* ma wartość 0,8 V, poniżej której sygnał wejściowy będzie uznany jako niski. Następna linia, na poziomie około 2,3 V, to wartość, która w przypadku wielu urządzeń 3,3 V będzie zinterpretowana jako stan logiczny równy 1. Linia 3,3 V pokazuje oczekiwaną przez Raspberry Pi wartość napięcia wejścia i wyjścia dla logicznego stanu wysokiego. Powyżej tej linii Pi może ulec zniszczeniu. Napięcie równe około 4,2 V to dla niektórych urządzeń 5 V spodziewana wartość stanu wysokiego (chociaż niektóre z nich za stan wysoki uznają nawet napięcie równe 2 V). Raspberry Pi potrzebuje urządzeń pomocniczych do komunikacji z takimi urządzeniami.

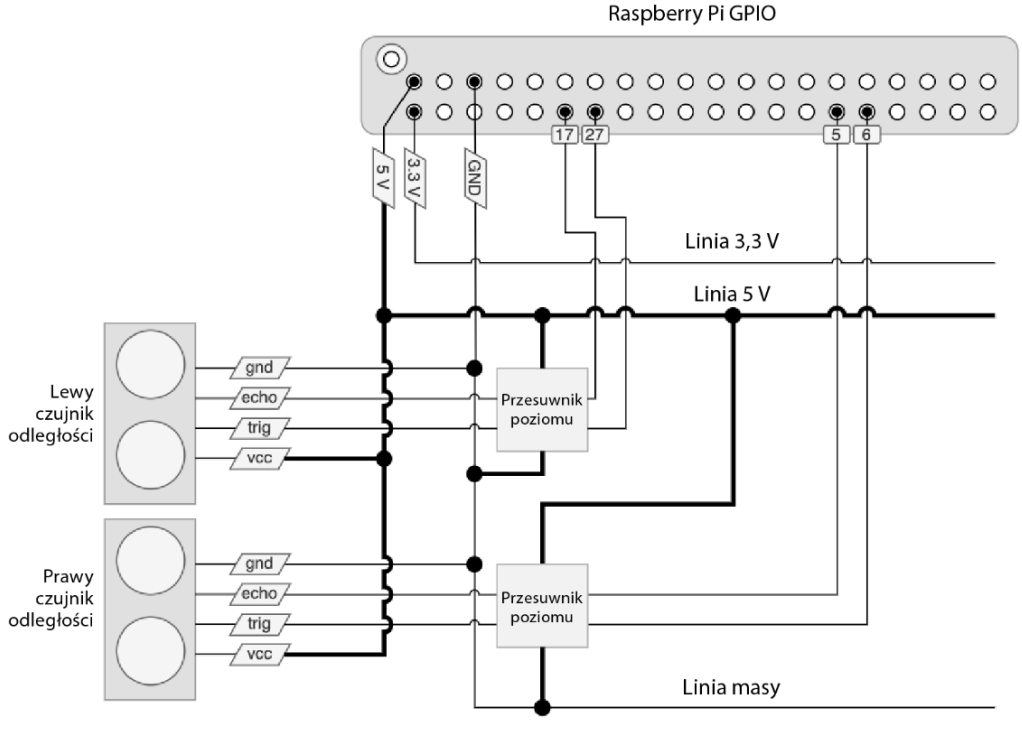
Na wykresie pokazano pięć słupków. Pierwszy opisany słupek ma wartość 0, co oznacza bezsprzeczny stan niski dla wszystkich typów urządzeń. Następny słupek, sięgający 3,3 V, to wyraźny stan wysoki dla Raspberry Pi, ale jest to wartość niższa niż 4,2 V, więc niektóre urządzenia 5 V jej nie rozpoznają. Słupek opisany jako *Niejasny* ma wartość 1,8 V, która znajduje się pomiędzy wartościami progowymi dla stanu wysokiego i niskiego. Dla urządzeń zarówno 3,3 V, jak i 5 V jest to wartość niejednoznaczna i powinno się jej unikać. Słupek oznaczony jako *Niewyraźny stan logiczny 1* jest ponad wartością graniczną, ale tylko trochę, więc ten stan może być mylnie interpretowany przez urządzenia 3,3 V i zwracać dziwne rezultaty. Ostatni słupek ma wartość 5 V, czyli napięcie wyjściowe urządzeń 5 V. Takie urządzenia nie mogą być podłączane do Raspberry Pi bez przesuwnika poziomu napięcia, gdyż zniszczą Pi.

Słupki 1,8 V i 2,3 V są bardzo blisko wartości progowych i mogą powodować, że dane odbierane z wejścia będą miały losową wartość. Unikaj wartości znajdujących się pomiędzy wartościami progowymi. Napięcie 3 V jest akceptowalne, ale unikaj 1,5 V, gdyż taka wartość jest niejasna.

> Przyłożenie napięcia wyższego niż 3,3 V do Raspberry Pi może je zniszczyć. Nie podłączaj urządzeń 5 V bez przesuwnika poziomu napięć.

Jeśli chcesz użyć urządzenia 5 V, potrzebujesz dodatkowej elektroniki, aby Raspberry Pi mogło się z nim komunikować. Wiąże się to z dodatkowymi przewodami i częściami, co wpływa na koszt, poziom złożoności i rozmiar robota.

Rysunek 8.5 pokazuje schemat podłączeniowy robota, który używa czujników HC-SR04 5V wymagających zastosowania przesuwników poziomu. W górnej części schematu znajdują się piny GPIO Raspberry Pi. Z trzech pinów zaznaczonych po lewej stronie wychodzą linie 5 V, 3,3 V (oznaczane jako 3v3) i masy (GND). Linie 3,3 V i 5 V biegną pod złączem GPIO.



Rysunek 8.5. Podłączenie czujników HC-SR04 do przesuwników poziomu

Poniżej linii zasilania (lub szyn zasilania) znajdują się dwa przesuwniki poziomu. Z ich prawej strony wychodzą połączenia do pinów GPIO 5, 6, 17 i 27 na Raspberry Pi. W tego typu schemacie czarne kropki oznaczają połączenia, a linie, które się nie łączą, mają kształt mostków.

Na dole schematu biegnie linia masy wychodząca z pinu masy (GND). Dodatkowe urządzenia zawsze potrzebują dostępu do masy.

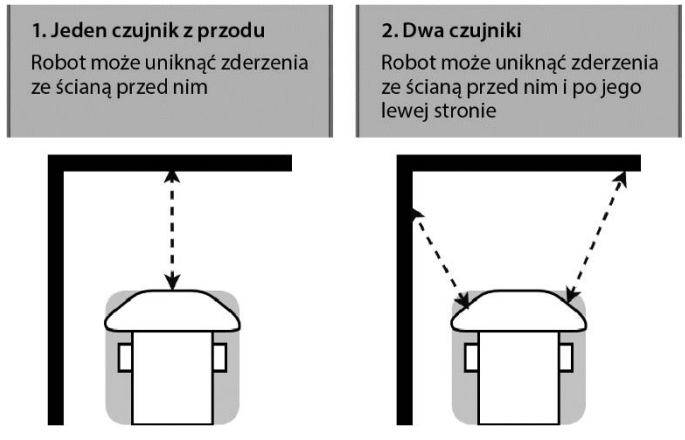
W lewej części schematu widać dwa czujniki odległości z połączeniem do 5 V i GND. Każdy czujnik ma piny *echo* (wejście sygnału odbitego) i *trig* (wyjście sygnału), które są podłączone do przesuwników poziomu. Nietrudno zauważyć, że dodanie kolejnych czujników, które także wymagałyby przesuwników, istotnie zwiększyłoby poziom skomplikowania obwodu.

Na szczęście są dostępne inne opcje. Zawsze, gdy jest to możliwe, warto stosować urządzenia 3,3 V lub urządzenia, które używają napięcia zasilania dla stanu wysokiego. Kupując elektronikę dla robota, weź pod uwagę napięcie używane przez główny kontroler (na przykład Raspberry Pi) i upewnij się, że wybrane urządzenie obsługuje takie napięcie.

Jest kilka odpowiedników czujnika HC-SR04, które mają taką możliwość. Modele HC-SR04P, RCWL-1601 i Adafruit 4007 mają napięcie wyjściowe równe 3,3 V i można je podłączyć bezpośrednio do Raspberry Pi.

Dlaczego dwa czujniki?

Dzięki dwóm czujnikom możemy wykryć, z której strony przeszkoda znajduje się bliżej. W ten sposób robot potrafi określić, gdzie są otwarte przestrzenie, i poruszać się w ich stronę. Na rysunku 8.6 pokazano, jak to działa.



Rysunek 8.6. Zastosowanie dwóch czujników

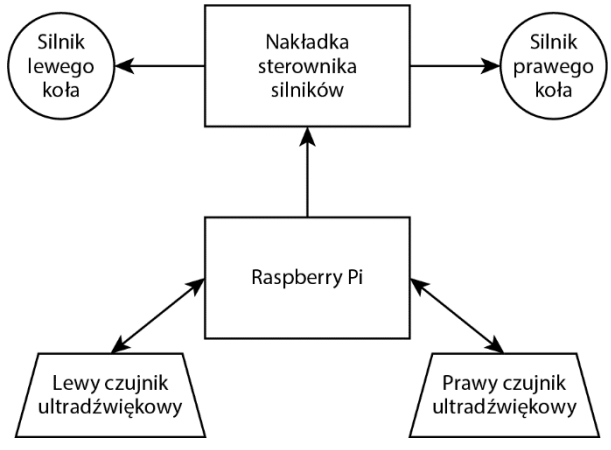
Na rysunku 8.6 drugi robot potrafi podejmować bardziej interesujące decyzje dzięki większej liczbie danych ze świata.

Biorąc pod uwagę wszystkie te opcje, polecam modele takie jak HC-SR04P/RCWL-1601 czy Adafruit 4007, gdyż są tanie i łatwo dodać dwa czujniki lub ich większą liczbę.

Poznaliśmy kilka rodzajów czujników odległości i omówiliśmy wszystkie za i przeciw w kontekście naszego robota. Dowiedziałeś się o poziomach napięcia i o tym, dlaczego napięcie jest takie ważne, jeśli chodzi o wybór części elektronicznych dla robota. Zastanawialiśmy się również, ile czujników zastosować i gdzie je zamontować. Teraz zobaczmy, jak je dodać.

Podłączanie czujnika ultradźwiękowego i odczytywanie z niego danych

Najpierw powinniśmy zamontować czujniki i podłączyć je do robota. Następnie napiszemy prosty kod testowy, na którym będzie się opierać kod zachowania, które utworzymy w następnym podrozdziale. Natomiast po tym podrozdziale schemat blokowy budowy robota powinien wyglądać jak ten pokazany na rysunku 8.7.



Rysunek 8.7. Schemat blokowy robota z czujnikami ultradźwiękowymi

Rysunek 8.7 przedstawia rozbudowany o lewy i prawy czujnik ultradźwiękowy schemat z rysunku 6.33 z rozdziału 6. „Podstawy budowania robota — koła, zasilanie i połączenia". Oba czujniki są połączone z Raspberry Pi dwukierunkowymi strzałkami, ponieważ Raspberry Pi wysyła do nich żądanie pomiaru, a następnie je z nich odczytuje. Zamontujmy czujniki na podwoziu robota.

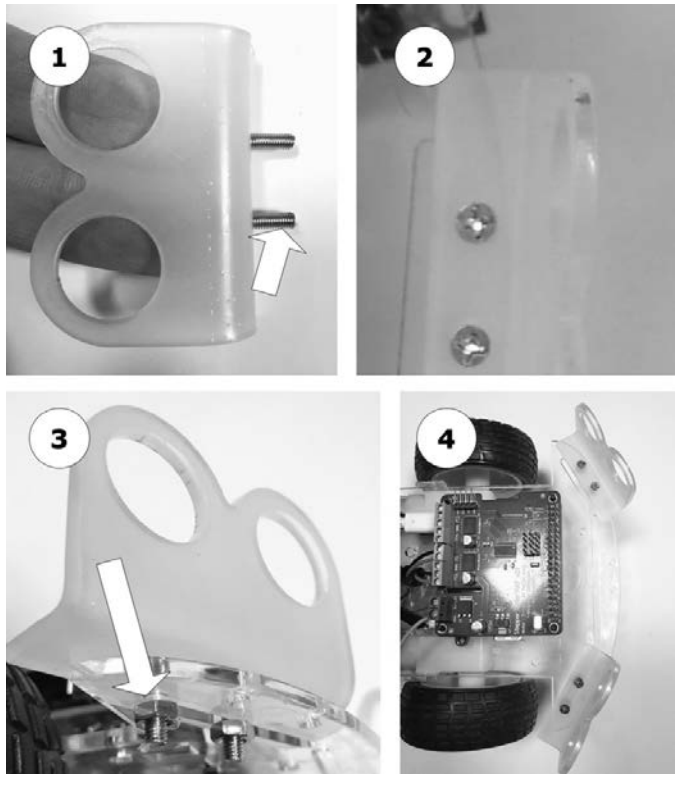
Montaż czujników

W „Wymaganiach technicznych" dla tego rozdziału wymieniłem wspornik dla HC-SR04. Pomimo że można zaprojektować i zrobić taki wspornik własnoręcznie, rozsądniej jest użyć gotowego. Rysunek 8.8 przedstawia stosowane przeze mnie wsporniki.

Rysunek 8.8. Wsporniki czujnika ultradźwiękowego HC-SR04 ze śrubkami i nakrętkami

Wybrane przeze mnie wsporniki są bardzo łatwe w montażu, pod warunkiem że masz podwozie podobne do mojego i ma ono otwory, do których można je przykręcić.

Aby zamontować wspornik czujnika, wykonaj poniższe kroki, wzorując się na rysunku 8.9.



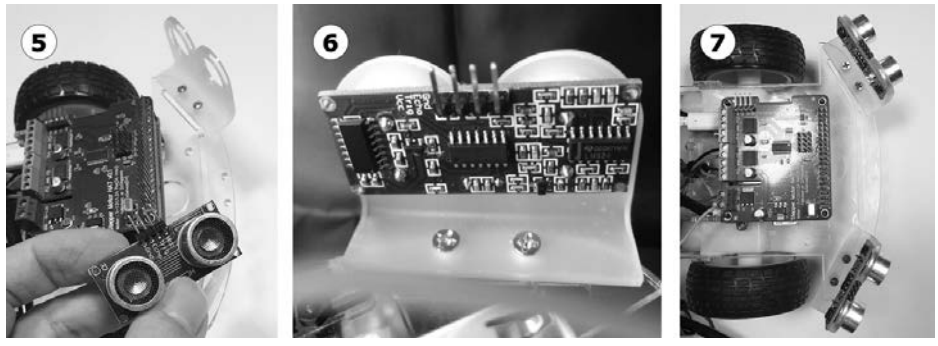
Rysunek 8.9. Montaż wspornika czujnika krok po kroku

1. Włóż dwie śrubki do otworów we wsporniku.
2. Wystające części śrubek umieszczonych we wsporniku włóż do otworów w platformie podwozia.
3. Nałóż od spodu nakrętkę na każdą ze śrubek i dokręć. Zrób to samo z drugim wspornikiem.
4. Robot z zamontowanymi dwoma wspornikami powinien wyglądać tak.

   Na rysunku 8.10 pokazano, jak wsadzić czujniki do wsporników.

5. Przyjrzyj się czujnikowi. Nadajnik i odbiornik — dwa okrągłe elementy z siatką na górze dobrze się wpasują w duże otwory wspornika.
6. Czujniki odległości mogą być po prostu wciśnięte do wspornika, gdyż będą dobrze przez niego przytrzymywane. Piny czujnika powinny być skierowane w górę.

7. Po włożeniu obu czujników robot powinien wyglądać jak ten pokazany na rysunku 8.10(7).



Rysunek 8.10. Wkładanie czujników do wsporników

Właśnie przymocowałeś czujniki do podwozia. Zanim je podłączymy, odejdziemy na chwilę od wykonywania głównego planu i dodamy bardzo przydatny włącznik zasilania.

Dodawanie przełącznika zasilania

Zanim ponownie włączymy robota, dodajmy przełącznik zasilania silników. Jest to wygodniejsze niż przykręcanie i odkręcanie przewodu masy baterii od złącza za każdym razem. Zobaczymy, jak to zrobić w trzech prostych krokach.

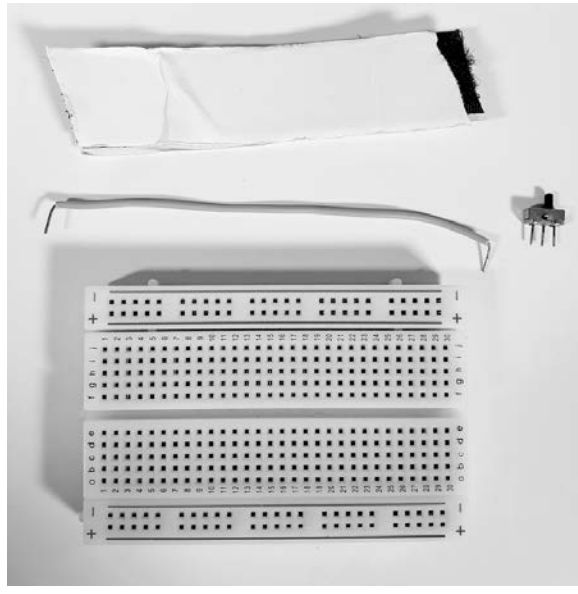
1. Upewnij się, że masz przygotowane elementy widoczne na rysunku 8.11: płytkę stykową, rzep, przełącznik SPDT, który można wpiąć do płytki stykowej, oraz przewód o średnicy 0,643 mm (22 AWG) i długości ok. 15 cm.
2. Teraz za pomocą dwóch rzepów przymocuj płytkę stykową na banku energii, tak jak to widać na rysunku 8.12. Dzięki rzepom płytka będzie trzymać się mocno i będzie można ją łatwo zdjąć, jeśli zajdzie taka potrzeba.

Po zamontowaniu płytki stykowej możemy dodać przełącznik.

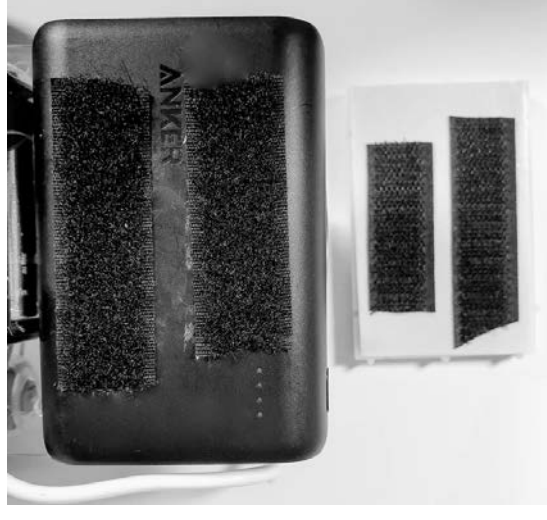
Rysunek 8.13 pokazuje szczegółowo, jak podłączyć przełącznik.

Rysunek 8.13 przedstawia schemat obwodu, zbliżenie na płytkę stykową oraz propozycję podłączenia przełącznika do robota. Przyjrzyjmy się temu bliżej.

1. Jest to schemat obwodu z bateriami, przełącznikiem i złączami zasilania silników. Na górze znajdują się wyprowadzenia *zasilania silników*. Od dodatniego styku (+) po lewej stronie baterii (która jest przedstawiona za pomocą naprzemiennych grubych i cienkich linii) biegnie przewód. Dolne wyprowadzenie baterii to strona ujemna, od której biegnie przewód do przełącznika pokazanego w prawej części schematu. Górna część przełącznika jest podłączona przewodem do ujemnej strony (–) *zasilania silników*. To jest ważny schemat podczas łączenia ze sobą wszystkich elementów.

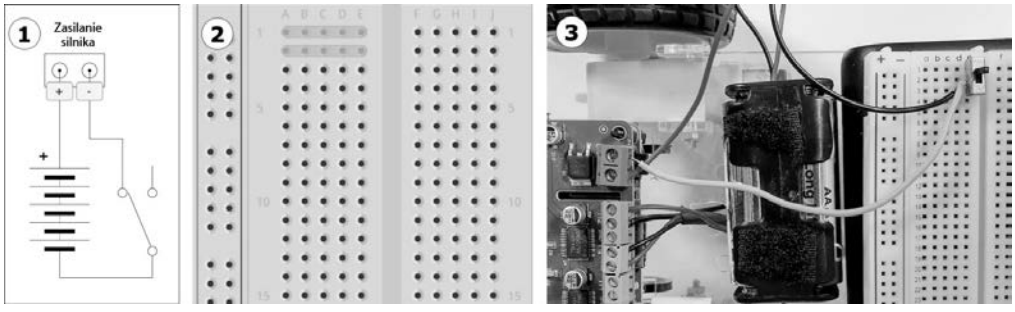


Rysunek 8.11. Części potrzebne do dodania przełącznika zasilania



Rysunek 8.12. Przyklejenie rzepów

2. Zanim fizycznie podłączymy przełącznik, warto wcześniej omówić wiersze płytki stykowej. Rysunek 8.13(2) przedstawia zbliżenie na płytkę stykową z zaznaczonymi na zielono dwoma wierszami. Zielone linie pokazują, że wiersz składa się z pięciu styków. Płytka stykowa jest podzielona na dwie grupy wierszy (ponumerowanych od 1 do 30) złożonych z pięciu otworów (styków). Pośrodku znajduje się wgłębienie, które te grupy oddziela.



Rysunek 8.13. Podłączanie przełącznika

3. Płytka stykowa pomoże fizycznie połączyć ze sobą podzespoły. Nie będzie to dokładne odzwierciedlenie schematu. Po lewej stronie widać sterownik silników, gdzie czerwony przewód wychodzący z baterii, ich dodatniej strony, wchodzi do jego złącza dodatniego (+ lub VIN). Baterie są pośrodku. Czarny przewód wychodzący z baterii jest wpięty w wierszu 3., w kolumnie *d*. W kolumnie *e* został umieszczony przełącznik, którego nóżki zostały wpięte w otworach w wierszach 1., 2. i 3. Pomarańczowy przewód 22 AWG biegnie z wiersza 2. do złącza śrubowego GND. Przesunięcie przełącznika spowoduje włączenie zasilania silników.

Właśnie dodaliśmy naszemu robotowi przełącznik dla baterii, dzięki czemu możemy włączać i wyłączać zasilanie bez użycia wkrętaka. Zaraz wykorzystamy tę samą płytkę stykową do podłączenia czujników odległości.

Podłączanie czujników odległości

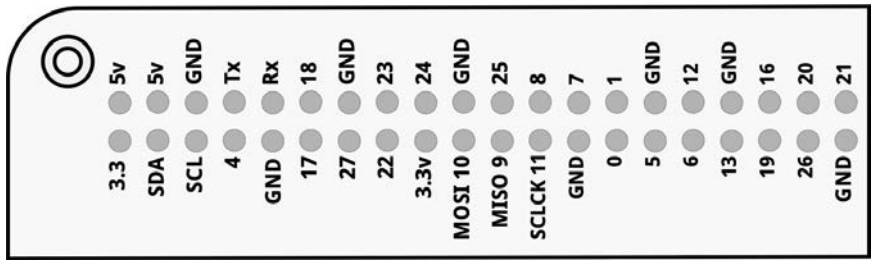
Każdy ultradźwiękowy czujnik ma cztery wyprowadzenia:

- pin wyzwalający (ang. *trigger*), który służy do wysłania żądania pomiaru;
- pin odbierający (ang. *echo*), który czeka na powrót odbitego sygnału;
- pin VCC, który powinien być podłączony do 3,3 V;
- pin masy (*GND*).

Zanim przejdziesz dalej, upewnij się, że robot jest wyłączony (zarówno zasilanie silników, jak i Raspberry Pi). Piny wyzwalający i odbierający należy podłączyć do pinów GPIO na Pi.

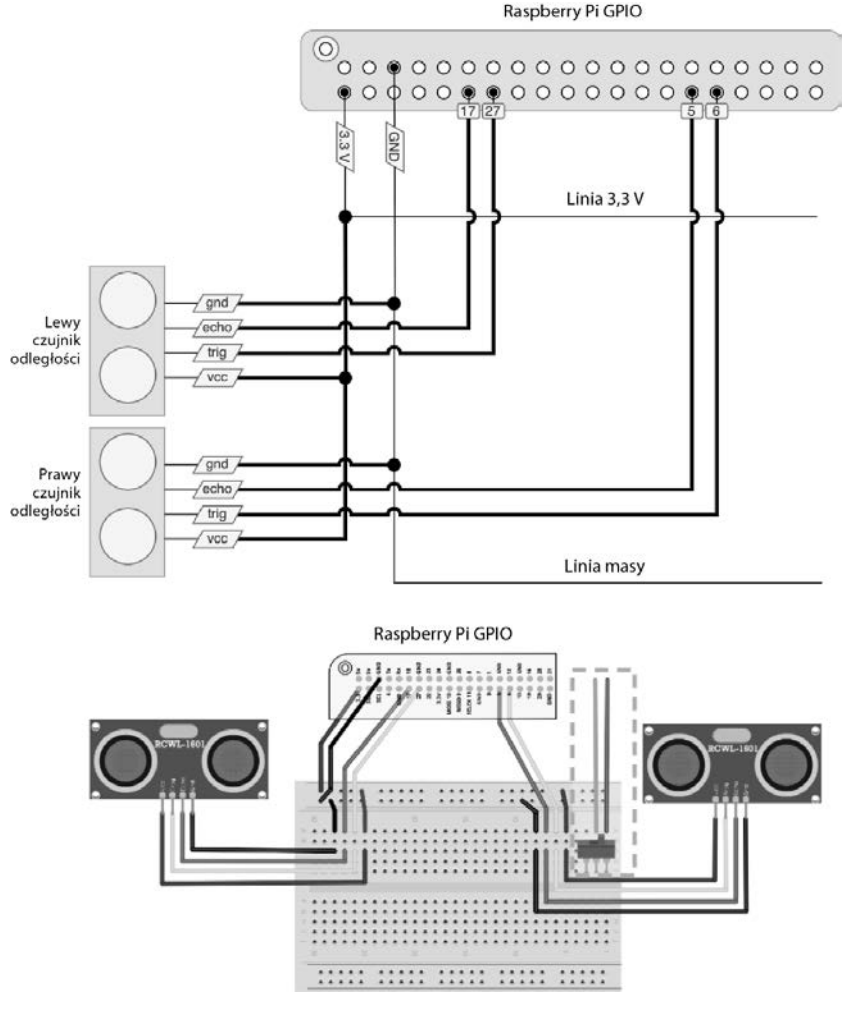
Rysunek 8.14 pokazuje zbliżenie na złącze GPIO Raspberry Pi, co pomoże Ci wykonać połączenia.

Na rysunku 8.14 pokazano schemat złącza GPIO znajdującego się w górnej części Raspberry Pi, które składa się z 40 pinów ułożonych w dwóch rzędach. Wiele robotów i gadżetów z niego korzysta. Numery i opisy pinów nie zostały umieszczone na Raspberry Pi, ale ten schemat pomoże Ci je znaleźć.



Rysunek 8.14. Piny GPIO Raspberry Pi

Do podłączenia czujników do złącza GPIO wykorzystamy płytkę stykową. Na rysunku 8.15 pokazano potrzebne połączenia.



Rysunek 8.15. Schemat podłączeniowy czujników

Aby wykonać połączenia między złączem GPIO i płytką stykową oraz między płytką stykową a czujnikami, potrzebujesz przewodów męsko-żeńskich. Do wykonania połączeń w obrębie płytki stykowej (są tylko cztery takie) wykorzystano krótkie zworki. Rysunek 8.15 przedstawia schemat obwodu (górna część) i propozycję połączeń na płytce stykowej (dolna część rysunku).

Aby podłączyć czujniki, wykonaj poniżej podane kroki, wzorując się na rysunku 8.15.

1. Zacznij od połączeń zasilania. Pin 3,3 V Raspberry Pi (często oznaczany na schematach jako *3v3*) należy podłączyć do szyny oznaczonej czerwoną linią na płytce stykowej. Możemy użyć tej samej czerwonej *szyny* dla innych połączeń, które wymagają napięcia 3,3 V.
2. Pin masy (*GND*) Raspberry Pi należy podłączyć do szyny oznaczonej czarną lub niebieską linią na płytce stykowej. Możemy wykorzystać tę *szynę* dla urządzeń, które wymagają połączenia z masą.
3. Dla każdej ze stron oderwij pasek składający się z czterech przewodów męsko-żeńskich.
4. Na lewym czujniku sprawdź, który z pinów to *VCC*, *trig*, *echo* i *GND*. Warto połączyć je z płytką stykową za pomocą czterech zgrupowanych ze sobą przewodów. Weź cztery przewody żeńsko-męskie (jeśli to możliwe, w postaci paska), podłącz je do czujnika, a następnie wepnij do pytki stykowej.
5. Na płytce stykowej za pomocą zworek podłącz masę do niebieskiej szyny, a napięcie zasilania do czerwonej linii.
6. Teraz za pomocą przewodów połącz piny *echo* i *trig* z pinami GPIO Raspberry Pi.

> W zależności od tego, gdzie umieściłeś płytkę stykową, przewody przeznaczone do podłączenia do niej czujników mogą okazać się za krótkie. Jeśli tak jest w Twoim przypadku, połącz ze sobą dwa przewody męsko-żeńskie i zabezpiecz miejsce połączenia za pomocą taśmy izolacyjnej.

Dla utrzymania porządku lubię zwijać przewody. Nie jest to obowiązkowe, ale pomaga zmniejszyć nieco ich gmatwaninę.

Zanim przejdziesz dalej, upewnij się, proszę, że wszystkie połączenia zostały poprawnie wykonane. Właśnie wzbogaciłeś swojego robota o czujniki odległości, lecz aby je przetestować i ich używać, trzeba przygotować odpowiednie oprogramowanie.

Instalacja bibliotek Pythona do komunikacji z czujnikiem

Ażeby móc korzystać z czujnika lub innego sprzętu podłączanego do złącza GPIO, potrzebujesz biblioteki Pythona. Użyjmy biblioteki *GPIOZero*, która została napisana, aby ułatwić komunikację z tego typu urządzeniami.

```
$ pip3 install RPi.GPIO gpiozero
```

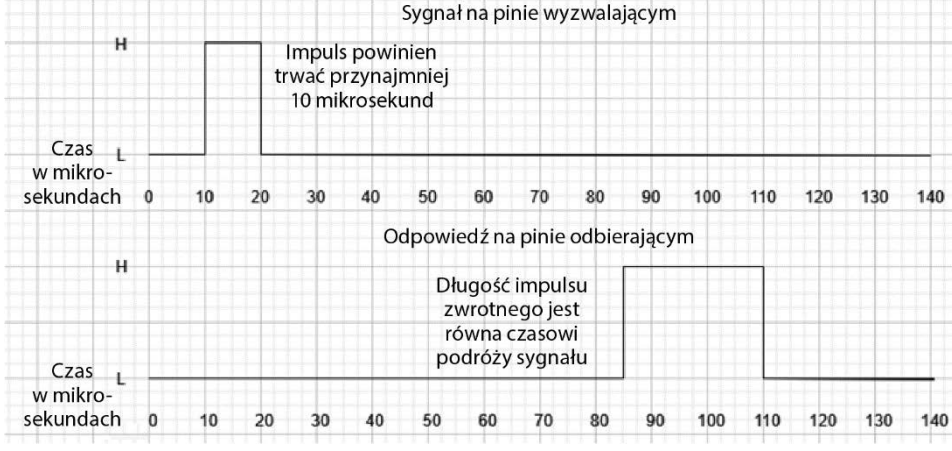
Gdy już mamy zainstalowaną bibliotekę, możemy napisać testowy kod.

Odczytywanie odległości z czujnika ultradźwiękowego

Przed napisaniem kodu dla czujnika odległości warto zrozumieć jego sposób działania. Jak już wcześniej wspomniano, zasada działania opiera się na wysyłaniu sygnałów dźwiękowych, które odbijają się od przedmiotów i wracają do czujnika, który mierzy czas, jaki sygnał potrzebował na powrót.

Kod uruchamiany na Raspberry Pi wysyła impuls elektryczny do pinu **wyzwalającego** (*trig*), sygnalizując w ten sposób żądanie pomiaru odległości. W odpowiedzi czujnik wysyła sygnał dźwiękowy i mierzy czas jego powrotu. Pin **odbierający** (*echo*) wysyła odpowiedź do Pi w postaci impulsu, którego długość odpowiada czasowi podróży sygnału dźwiękowego.

Wykres na rysunku 8.16 pokazuje pomiary czasowe.



Rysunek 8.16. Pomiary czasowe impulsu i odpowiedzi ultradźwiękowego czujnika odległości

Biblioteka *GPIOZero* potrafi mierzyć czas tych impulsów i przeliczać go na odległość, którą możemy wykorzystać w naszym kodzie.

Urządzenie może nie zwrócić odczytu na czas, jeśli odbity sygnał nie wrócił wystarczająco szybko. Możliwe, że obiekt będzie poza zakresem czujnika lub coś stłumi sygnał dźwiękowy.

Podobnie jak w przypadku klasy sterującej silnikami, powinniśmy stosować komentarze i nazwy, które dobrze oddają rolę danego elementu kodu, co ułatwi nam omawianie tej części skryptu. Nazwałem ten plik *test_distance_sensors.py* (test czujników odległości).

1. Na początku zaimportujemy biblioteki *time* i *DistanceSensor*:

```
import time
from gpiozero import DistanceSensor
```

2. Następnie zainicjalizujemy czujniki. Dodałem wyrażenie `print`, aby pokazać, co się dzieje. W tych linijkach tworzymy obiekt biblioteki dla każdego czujnika odległości,

podając piny, do których są podłączone. Upewnij się, że podane niżej linijki pasują do Twoich połączeń:

```
print("Przygotowanie pinów GPIO")
sensor_l = DistanceSensor(echo=17, trigger=27, queue_len=2)
sensor_r = DistanceSensor(echo=5,  trigger=6, queue_len=2)
```

Zwróć uwagę na dodatkowy argument `queue_len`. Biblioteka *GPIOZero* stara się zebrać 30 odczytów z czujników, zanim zwróci odpowiedź, dzięki czemu działa płynnie, ale jest mniej responsywna. Jednak my chcemy, aby nasz robot szybko reagował, dlatego zmniejszymy liczbę odczytów do dwóch.

3. Następnie w pętli jest wykonywany test, dopóki go nie przerwiemy.

```
while True:
```

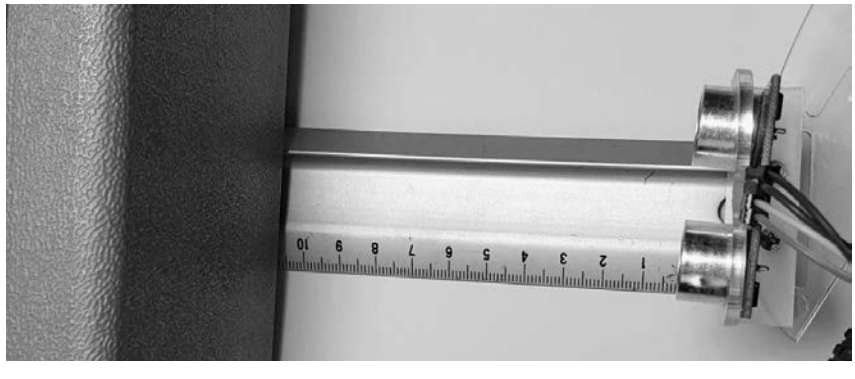
4. Potem wyświetlamy odległość zmierzoną przez nasze czujniki. `.distance` jest właściwością, podobnie jak `.count`, którą widzieliśmy już wcześniej w tej książce, w systemie diod LED. Czujniki cały czas aktualizują jej wartość. Mnożymy ją przez 100, gdyż odległość w bibliotece *GPIOZero* jest podawana w metrach.

```
print("Lewy: {l}, Prawy: {r}".format(
    l=sensor_l.distance * 100,
    r=sensor_r.distance * 100))
```

5. Mała pauza w pętli zapobiega zalewowi danych wejściowych i ogranicza liczbę iteracji.

```
time.sleep(0.1)
```

6. Teraz możesz włączyć Raspberry Pi i przesłać napisany właśnie kod.
7. Umieść przedmiot gdziekolwiek w odległości od 4 cm do 1 m od czujnika, tak jak pokazano na rysunku 8.17.



Rysunek 8.17. Czujnik odległości z umieszczonym przed nim przedmiotem

Na rysunku 8.17 widać przedmiot znajdujący się około 10,5 cm od czujnika. Jest to mała skrzynka na narzędzia. Ważne jest to, że jest sztywna i niewykonana z tkaniny.

8. Uruchom kod na Pi za pomocą polecenia `python3 test_distance_sensors.py`. Gdy będziesz poruszać obiektem przed czujnikami, Pi powinno pokazywać odległości:

```
pi@myrobot:~ $ python3 test_distance_sensors.py
Przygotowanie pinów GPIO
Lewy: 6.565688483970461, Prawy: 10.483658125707734
Lewy: 5.200715097982538, Prawy: 11.58136928065528
```

9. Ponieważ kod umieszczony jest w pętli, musisz nacisnąć *Ctr+C*, aby zatrzymać program.
10. Zobaczysz liczby z wieloma miejscami po przecinku, co nie jest tutaj zbyt pomocne. Po pierwsze, nasze czujniki raczej nie są tak dokładne, a po drugie, nasz robot nie potrzebuje aż tak precyzyjnych pomiarów, aby podejmować decyzje. Możemy zmienić trochę wyrażenie `print` w pętli, aby było bardziej czytelne:

    ```
    print("Lewy: {l:.2f}, Prawy: {r:.2f}".format(
        l=sensor_l.distance * 100,
        r=sensor_r.distance * 100))
    ```

    `:.2f` zmienia sposób wyświetlania tekstu i zawęża liczby zmiennoprzecinkowe do dwóch miejsc dziesiętnych. Ponieważ wyświetlanie danych jest pomocne przy ewentualnym usuwaniu problemów, warto wiedzieć, jak je dostosować do swoich potrzeb.
11. Uruchomienie kodu z tą zmianą da następujący rezultat:

    ```
    pi@myrobot:~ $ python3 test_distance_sensors.py
    Przygotowanie pinów GPIO
    Lewy: 6.56, Prawy: 10.48
    Lewy: 5.20, Prawy: 11.58
    ```

Pokazałeś, że czujnik odległości działa. Dowiedziałeś się również, jak dostosować wyświetlanie odczytów z czujników, aby ułatwić wykrywanie i usuwanie błędów, co będziesz robić bardzo często podczas tworzenia robotów. Aby upewnić się, że do tego momentu wszystko działa, rozwiąż ewentualne problemy, które mogłeś napotkać.

Wykrywanie i rozwiązywanie problemów

Jeśli Twoje czujniki nie działają tak, jak powinny, spróbuj wykonać poniżej wymienione kroki:

- Czy jakiś przewód jest gorący? Przytrzymaj między kciukiem a palcem wskazującym przewody biegnące od czujnika. **Nic nie powinno być gorące ani się nagrzewać!** Jeśli tak, usuń baterie, wyłącz Raspberry Pi i dokładnie sprawdź wszystkie połączenia z rysunkiem 8.12.
- Jeśli błędy są związane ze składnią, sprawdź swój kod z tym podanym w książce lub w kodach do pobrania. Powinieneś zainstalować bibliotekę za pomocą komendy `pip3`, a skrypt uruchamiać przy użyciu polecenia `python3`.
- Jeśli wciąż zmagasz się z błędami lub niepoprawnymi wartościami, sprawdź kod i wcięcia.
- Jeśli zmierzona odległość za każdym razem jest równa 0 lub czujnik nie zwraca żadnych wartości, możliwe, że podłączyłeś pin wyzwalający i odbierający na odwrót. Zamień numery tych pinów w kodzie i ponownie przetestuj czujniki. **Nie** zamieniaj przewodów przy włączonym Pi! Najpierw zamień numery pinów dla jednego czujnika, a następnie dla drugiego.

- Jeżeli dalej nie otrzymujesz żadnych wyników, upewnij się, że kupiłeś czujniki 3,3 V. Model HC-SR04 nie będzie działał z Raspberry Pi bez dodatkowego sprzętu.
- Jeśli odczytane wartości są spoza zakresu lub są zbyt duże, upewnij się, że powierzchnia badanego przedmiotu jest twarda. Pomiar w przypadku miękkich powierzchni, takich jak ubrania, zasłony lub dłoń, nie jest tak wiarygodny jak dla szkła, drewna, metalu czy plastiku. Ściana również zwraca poprawne odczyty!
- Inną przyczyną niepoprawnych wartości jest zbyt mała powierzchnia obiektu. Upewnij się, że Twoja powierzchnia jest dość szeroka. Wszystko, co jest mniejsze niż 5 cm^2, może być trudne do zmierzenia.
- W ostateczności, jeśli jeden czujnik działa poprawnie, a drugi nie, możliwe, że urządzenie jest wadliwe. Spróbuj zamienić czujniki miejscami. Jeśli otrzymany wynik jest różny, to może oznaczać, że czujnik jest zepsuty. Jeśli natomiast wynik jest taki sam, problem leży po stronie połączeń lub kodu.

Właśnie rozwiązałeś ewentualne problemy związane z czujnikami odległości i upewniłeś się, że działają. Wyświetliłeś na ekranie odczyty, aby pokazać, że czujniki funkcjonują, i zmierzyłeś odległość dla kilku przedmiotów. Teraz pójdźmy krok dalej i napiszmy skrypt w celu omijania przeszkód.

Unikanie ścian — skrypt omijania przeszkód

Po sprawdzeniu obu czujników możemy dodać zachowanie polegające na unikaniu przeszkód do naszej klasy `Robot`. To zachowanie będzie polegało na odczytaniu danych z czujników, a następnie wybraniu odpowiedniej reakcji.

Dodawanie czujników do klasy Robot

Zanim będziemy mogli używać czujników w naszym zachowaniu, musimy je dodać do klasy `Robot` i przypisać odpowiednie piny dla każdej strony. Dzięki temu, jeśli zmienią się piny lub nawet cały interfejs czujników, kod zachowania będzie mógł pozostać bez zmian.

1. Aby móc korzystać z obiektu `DistanceSensor`, musimy go zaimportować z biblioteki *GPIOZero*. Nowy kod został zaznaczony pogrubioną czcionką.

```
from Raspi_MotorHAT import Raspi_MotorHAT
from gpiozero import DistanceSensor
```

2. Następnie w klasie `Robot` tworzymy dwie instancje klasy `DistanceSensor`, po jednej dla każdej strony. Musimy dodać to do konstruktora klasy `Robot`. Użyjemy tych samych pinów i liczby odczytów co w kodzie testowym.

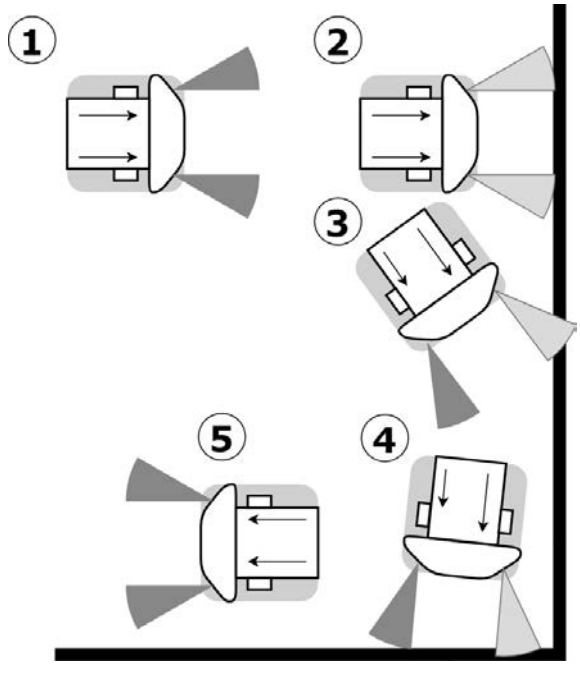
```
class Robot:
    def __init__(self, motorhat_addr=0x6f):
        # Inicjalizacja nakładki sterownika silników znajdującej się pod podanym adresem
```

```
        self._mh = Raspi_MotorHAT(addr=motorhat_addr)

        # Pobranie zmiennej lokalnej dla każdego silnika
        self.left_motor = self._mh.getMotor(1)
        self.right_motor  = self._mh.getMotor(2)

        # Inicjalizacja czujników
        self.left_distance_sensor = DistanceSensor(echo=17, trigger=27,
        ↪queue_len=2)
        self.right_distance_sensor = DistanceSensor(echo=5, trigger=6,
        ↪queue_len=2)
        # Upewnienie się, że silniki się zatrzymają, gdy program przestanie działać
        atexit.register(self.stop_all)
```

Dzięki dodaniu tych linijek do warstwy robota czujniki są dostępne w warstwie zachowań. Gdy utworzymy obiekt robota, czujniki będą mierzyć odległość. Stwórzmy zachowanie, które korzysta z tych odczytów.

Zachowania polegające na omijaniu przeszkód

Celem tego rozdziału jest stworzenie zachowania, które polega na omijaniu (większości) przeszkód przez poruszającego się robota. Wybrany przez nas rodzaj czujnika nie wykrywa mniejszych obiektów lub takich, które są pokryte miękką powłoką, na przykład przedmiotów obitych tapicerką. Podstawowe zachowanie, które chcemy osiągnąć, zostało zobrazowane na rysunku 8.18.

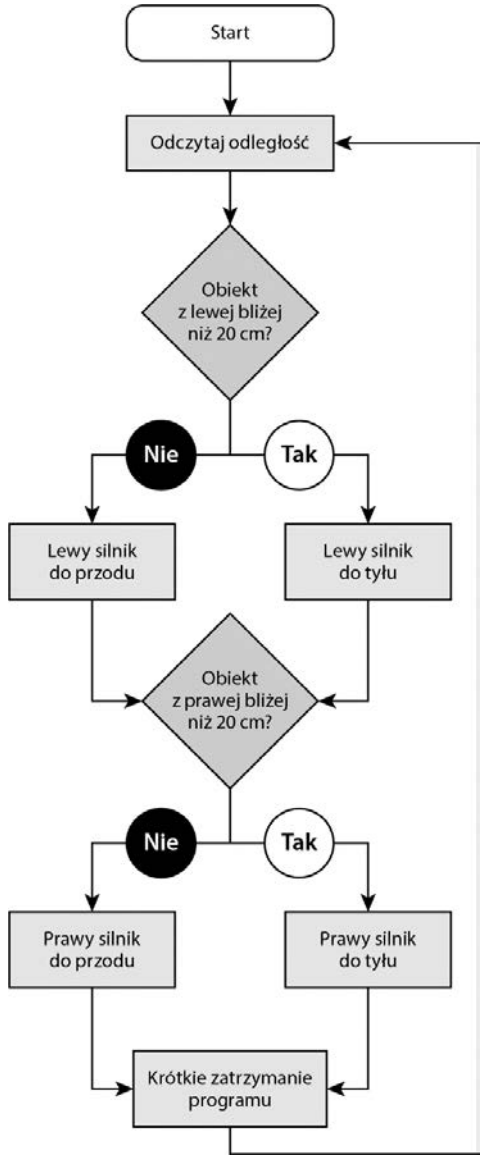


Rysunek 8.18. Podstawowa zasada unikania przeszkód

W naszym przykładzie (rysunek 8.18) robot wykrywa ścianę, skręca i jedzie dalej, zanim nie wykryje kolejnej ściany, przed którą również skręci. Możemy to wykorzystać w naszej pierwszej przymiarce do zachowania polegającego na unikaniu ścian.

Pierwsza próba omijania przeszkód

Aby łatwiej było nam zrozumieć to zadanie, na rysunku 8.19 pokazano jego schemat blokowy.



Rysunek 8.19. Schemat blokowy omijania przeszkód

Początek schematu blokowego pokazanego na rysunku 8.19 znajduje się u góry.

Schemat przedstawia pętlę, która wykonuje następujące zadania:

1. Znajdujący się po bloku *Start* blok *Odczytaj odległość* pobiera odczyty z obu czujników.
2. Sprawdzamy, czy odległość odczytana z lewego czujnika jest mniejsza niż 20 cm (rozsądna wartość progowa):
   a) Jeśli tak, lewy silnik będzie obracać się do tyłu, aby robot skręcił i odwrócił się od przeszkody.
   b) W przeciwnym razie lewy silnik powinien kręcić się do przodu.
3. Teraz sprawdzamy prawy czujnik i jeśli odczytana z niego odległość jest mniejsza niż 20 cm, prawy silnik będzie kręcił się do tyłu, a jeśli nie, to do przodu.
4. Program zatrzymuje się na krótką chwilę i ponownie wykonuje wyżej wymienione kroki.

Umieścimy tę pętle w metodzie `run`. W związku z tym musimy przygotować kilka rzeczy. Należy ustawić obrót i pochylenie na 0, aby nie blokowały czujników. Poniżej pokazany kod umieściłem w pliku *simple_avoid_behavior.py* (proste zachowanie unikające).

1. Najpierw zaimportuj klasę `Robot` i funkcję `sleep` w celu zatrzymania pętli przed kolejnym obiegiem.

```
from robot import Robot
from time import sleep
...
```

2. Poniżej pokazana klasa będzie podstawą dla naszego zachowania. W tym zachowaniu przechowywany jest obiekt robota. Została ustawiona prędkość, która może być zmieniona, aby robot jechał szybciej lub wolniej. Gdy będzie jechał zbyt szybko, będzie miał mniej czasu na reakcję.

```
...
class ObstacleAvoidingBehavior:
    """Proste unikanie przeszkód"""
    def __init__(self, the_robot):
        self.robot = the_robot
        self.speed = 60
        ...
```

3. Następująca metoda przypisuje każdemu silnikowi prędkość, która zależy od odległości zmierzonej przez czujnik. Gdy odległość jest mniejsza niż wartość progowa, robot odwraca się od przeszkody.

```
    ...
    def get_motor_speed(self, distance):
        """Ta metoda wybiera prędkość dla silnika w oparciu o odczyt z czujników."""
        if distance < 0.2:
            return -self.speed
        else:
            return self.speed
    ...
```

4. Metoda `run` jest najważniejsza, gdyż zawiera główną pętlę. W środku umieściliśmy obsługę obrotu i pochylania tak, aby czujniki nie były blokowane.

```
...
def run(self):
    self.robot.set_pan(0)
    self.robot.set_tilt(0)
```

5. Teraz zaczniemy pisać pętlę główną.

```
        while True:
            # Odczytanie odległości podanej w metrach
            left_distance = self.robot.left_distance_sensor.distance
            right_distance = self.robot.right_distance_sensor.distance
            ...
```

6. Następnie wyświetlimy odczyty w konsoli.

```
            ...
            print("Lewy: {l:.2f}, Prawy: {r:.2f}".format(l=left_distance,
r=right_distance))
            ...
```

7. Teraz odczytane wartości wykorzystamy z wcześniej napisaną metodą `get_motor_speed` i ustawimy prędkość każdego silnika.

```
            ...
            # Odczytanie odległości podanej w metrach
            left_speed = self.get_motor_speed(left_distance)
            self.robot.set_left(left_speed)
            right_speed = self.get_motor_speed(right_distance)
            self.robot.set_right(right_speed)
```

8. Jako że jest to pętla główna, przed jej kolejnym obiegiem zatrzymamy na chwilę program. Pod pętlą znajduje się inicjalizacja obiektu robota i zachowania omijania przeszkód oraz uruchomienie zachowania.

```
            ...
            # Krótkie zatrzymanie programu
            sleep(0.05)

bot = Robot()
behavior = ObstacleAvoidingBehavior(bot)
behavior.run()
```

Kod tego zachowania jest teraz gotowy i można go uruchomić. Nadszedł czas, aby go wypróbować. Do testów przygotuj kilka metrów kwadratowych podłogi. Nie umieszczaj tam przeszkód pokrytych materiałem lub cienkich obiektów, na przykład nóg krzesła. Ja do testów wykorzystałem segregatory i plastikowe pudełka na zabawki.

Wyślij napisany właśnie kod do robota i wypróbuj go. Robot powinien jechać prosto, dopóki nie napotka przeszkody, a następnie odwrócić się. Taka jest główna zasada działania. Możesz dostosować prędkości i wartości progowe, jeśli jest taka potrzeba. Jednak robot utyka w rogach i nie wie, co robić dalej.

Może nadszedł czas pomyśleć o lepszej strategii działania.

Bardziej zaawansowane omijanie przeszkód

Poprzednie zachowanie może doprowadzić do tego, że robot utknie. Okazuje się, że nie potrafi podjąć decyzji w przypadku niektórych przeszkód, a od czasu do czasu może nawet uderzać napotkane na swojej drodze obiekty. Może nie zatrzymać się na czas lub skręcić i wpaść na inną przeszkodę. Stwórzmy lepsze zachowanie, które będzie działać sprawniej.

Zatem co jest nie tak w naszej strategii działania? Otóż przeanalizujmy to z punktu widzenia czujnika znajdującego się najbliżej przeszkody i najdalej. Możemy ustalić prędkość silnika znajdującego się po stronie czujnika bliżej przeszkody i dalej oraz czas zatrzymania pętli głównej. Nasz kod wykorzysta czas zatrzymania przed następnym obiegiem pętli, co wpłynie na kąt odwrócenia się od przeszkody. Wprowadźmy kilka zmian w poprzednim kodzie.

1. Najpierw skopiuj plik *simple_avoid_behavior.py* i zmień jego nazwę na *avoid_behavior.py* (zachowanie unikające).
2. Nie będziesz potrzebować metody `get_motor_speed`, dlatego ją usuń. Zastąp ją funkcją o nazwie `get_speeds` pobierającą jeden argument, `nearest_distance`, którego wartość powinna być zawsze równa mniejszej odległości odczytanej z czujników.

```
...
    def get_speeds(self, nearest_distance):
        if nearest_distance >= 1.0:
            nearest_speed = self.speed
            furthest_speed = self.speed
            delay = 100
        elif nearest_distance > 0.5:
            nearest_speed = self.speed
            furthest_speed = self.speed * 0.8
            delay = 100
        elif nearest_distance > 0.2:
            nearest_speed = self.speed
            furthest_speed = self.speed * 0.6
            delay = 100
        elif nearest_distance > 0.1:
            nearest_speed = -self.speed * 0.4
            furthest_speed = -self.speed
            delay = 100
        else: # Kolizja
            nearest_speed = -self.speed
            furthest_speed = -self.speed
            delay = 250
        return nearest_speed, furthest_speed, delay
...
```

Te linijki służą do dostosowywania wartości. Podstawowa zasada polega na tym, że w zależności od odległości zmniejszamy prędkość silnika znajdującego się dalej od przeszkody, a jeśli robot bardzo się zbliży, odjedzie od napotkanego obiektu. Możemy kierować naszym robotem, opierając się na czasie zatrzymania programu i dzięki rozróżnieniu silników.

3. Większość linijek pozostałej części kodu jest taka sama. To jest funkcja `run`, którą już znasz:

```
...
def run(self):
     # Jazda do przodu
    self.robot.set_pan(0)
    self.robot.set_tilt(0)
    while True:
          # Odczytanie odległości podanej w metrach
        left_distance = self.robot.left_distance_sensor.distance
        right_distance = self.robot.right_distance_sensor.distance
          # Wyświetlenie odległości
        self.display_state(left_distance, right_distance)
        ...
```

4. Teraz robot za pomocą metody `get_speeds` określa, która z dwóch zmierzonych odległości jest mniejsza. Zauważ, że wyznaczamy minimum (`min`) z dwóch wartości. Metoda zwraca prędkości dla obu silników i czas, na jaki pętla ma się zatrzymać, a następnie wyświetla wartości tych zmiennych, abyśmy widzieli, co się dzieje.

```
            ...
              # Ustalenie prędkości silników na podstawie odległości
            nearest_speed, furthest_speed, delay =
self.get_speeds(min(left_distance,\ right_distance))
            print(f"Odległości: l {left_distance:.2f}, r {right_distance:.2f}.\
Prędkości: n: {nearest_speed}, f: {furthest_speed}. Pauza: {delay}")
            ...
```

Zastosowaliśmy interpolację łańcuchów znaków (ang. *f-string*), która jest skróconą wersją formatowania `.format` (którego używaliśmy wcześniej). Umieszczenie przedrostka w postaci litery *f* z przodu łańcucha umożliwia podanie zmiennych lokalnych w nawiasach klamrowych w samym łańcuchu znaków. Nadal możemy stosować `.2f` do określania liczby miejsc po przecinku.

5. Teraz sprawdzamy, z której strony przeszkoda znajduje się bliżej, z lewej czy z prawej, i odpowiednio ustawiamy prędkości silników.

```
            ...
              # Ustawienie prędkości silników
            if left_distance < right_distance:
                self.robot.set_left(nearest_speed)
                self.robot.set_right(furthest_speed)
            else:
                self.robot.set_right(nearest_speed)
                self.robot.set_left(furthest_speed)
            ...
```

6. Zamiast zatrzymać program przed następnym obiegiem pętli na stale ustaloną liczbę sekund, skrypt czeka przez czas podany w zmiennej `delay`. Zmienna ta przechowuje czas w milisekundach, więc musimy go pomnożyć, aby otrzymać sekundy.

```
            ...
              # Zatrzymanie programu na czas określony przez zmienną delay
            sleep(delay * 0.001)
...
```

7. Pozostała część kodu nie uległa zmianie. Cały kod znajdziesz w archiwum pobranym pod adresem *https://ftp.helion.pl/przyklady/jazar2.zip*, w folderze *r08*.

Po uruchomieniu tego kodu powinieneś zauważyć, że robot sprawniej unika przeszkód. Możliwe, że będziesz musiał dopracować czasy i wartości. Być może będziesz musiał odpowiednio zmienić wartości w blokach kodu ostatnich dwóch warunków, odpowiedzialnych za wycofywanie i skręcanie do tyłu. Jeśli robot nie jedzie do tyłu wystarczająco daleko, zwiększ czas, a jeżeli odjeżdża za daleko, zmniejsz czas.

Aczkolwiek to zachowanie ma wciąż wady. Robot nie tworzy mapy obszaru, po którym się porusza, i nie ma czujników z tyłu, więc podczas omijania przeszkód napotkanych z przodu może wpaść na obiekty znajdujące się z tyłu. Dodanie większej liczby czujników mogłoby rozwiązać kilka problemów. Jednak wciąż nie możemy stworzyć mapy, gdyż nasz robot nie ma czujników, które dokładnie zmierzyłyby, jak bardzo skręcił i jak daleko pojechał.

Podsumowanie

W tym rozdziale dodaliśmy czujniki do naszego robota. To ważny krok w stronę autonomicznego robota, który sam decyduje o swoim zachowaniu i reaguje w określony sposób na swoje otoczenie. Dowiedziałeś się, jak wzbogacić naszego robota o czujniki odległości, poznając przy tym ich różne dostępne rodzaje. Napisałeś kod, dzięki któremu zobaczyłeś, czy i jak działają czujniki. Następnie stworzyłeś zachowanie polegające na unikaniu ścian. Najpierw była to jego uproszczona wersja, ale często zawodna, więc dowiedziałeś się, jak ulepszyć ten system i napisać bardziej wyszukane zachowanie.

Dzięki temu doświadczeniu możesz zastanowić się, jak inne czujniki komunikowałyby się z Twoim robotem i jak wyglądałby prosty kod do ich obsługi. Możesz wyświetlać dane odczytane z czujników, więc usuwanie błędów jest łatwiejsze i dużo wygodniej stworzyć zachowanie, które sprawi, że robot będzie sam, w uproszczony sposób odnajdował drogę.

W następnym rozdziale wrócimy do omawiania poruszania się po określonej z góry ścieżce i linii prostej z wykorzystaniem enkodera, aby porównać to, jak nasz robot faktycznie się poruszył, z tym, co zakładaliśmy, dzięki czemu będziemy mogli dopracować skręcanie.

Ćwiczenia

1. Niektóre roboty radzą sobie tylko z jednym czujnikiem. Zastanów się, jak mógłby działać niezawodny system unikania przeszkód z jednym czujnikiem.
2. Później wykorzystamy mechanizm uchylno-obrotowy dla naszej kamery. Rozważ umieszczenie czujnika na takim mechanizmie i zastanów się, jak przekuć to na zachowanie.

3. Stworzone w tym rozdziale zachowanie może sprawiać, że nasz robot będzie wjeżdżał tyłem w przeszkody. Jak można rozwiązać ten problem? Może zrobisz projekt i spróbujesz to zbudować.

Lektura uzupełniająca

Więcej informacji znajdziesz w poniższych materiałach:

- Czujnik RCWL-1601 jest podobny do HC-SR04. Dokumentacja (w języku angielskim) tego drugiego czujnika zawiera przydatne informacje na temat jego zakresu. Znajdziesz ją na stronie *https://www.mouser.com/datasheet/2/813/HCSR04-1022824.pdf*.
- Na stronie internetowej ModMyPi znajdziesz samouczek (w języku angielskim), jak podłączyć czujnik HC-SR04 i jak przesuwać jego poziomy wejścia/wyjścia: *https://thepihut.com/blogs/raspberry-pi-tutorials/hc-sr04-ultrasonic-range-sensor--on-the-raspberry-pi*.
- Samouczki Raspberry Pi (w języku angielskim) omawiają budowę płytki stykowej i skrypt Pythona, który korzysta z biblioteki *RPi.GPIO* zamiast *GPIOZero*: *https://tutorials-raspberrypi.com/raspberry-pi-ultrasonic-sensor-hc-sr04/*.
- Zaczęliśmy korzystać z wielu pinów na Raspberry Pi. Aby upewnić się, których pinów należy użyć, polecam zajrzeć na stronę *https://pinout.xyz/*.
- Omówiliśmy pokrótce, jak wyświetlać dane potrzebne podczas znajdowania i usuwania ewentualnych błędów i jak można je dostosować do własnych potrzeb. Na stronie W3schools znajdziesz interaktywny przewodnik (w języku angielskim) na temat formatowania łańcuchów znaków w Pythonie: *https://www.w3schools.com/python/python_strings.asp*.
- Jest wiele artykułów naukowych traktujących o bardziej interesujących i zaawansowanych zachowaniach. Zachęcam do lektury artykułu (w języku angielskim) zatytułowanego *Simple, Real-Time Obstacle Avoidance Algorithm* (algorytm prostego unikania przeszkód w czasie rzeczywistym), który znajdziesz na stronie *http://www.wseas.us/e-library/conferences/2009/tenerife/CIMMACS/CIMMACS-03.pdf*.

Zbuduj i zaprogramuj inteligentnego robota!

Coraz więcej złożonych, powtarzalnych zadań powierzamy automatom. Inteligentny robot nigdy się nie znudzi, nie zmęczy i będzie cały czas pracował z zadaną prędkością. Zapewnia nam to odpowiednią wydajność i bardzo dużą dokładność wykonywanych czynności. Oczywiście, aby osiągnąć te korzyści, najpierw trzeba robota zbudować i zaprogramować. Warto spróbować własnych sił w tej dziedzinie. Wiedza o programowaniu autonomicznych robotów jest coraz bardziej ceniona na rynku pracy, a samo budowanie robotów i ich programowanie może być niesamowicie interesującym hobby!

Ta książka stanowi przystępne wprowadzenie do świata projektantów i budowniczych robotów. Dzięki niej dowiesz się, jak wybrać potrzebne podzespoły, jak je ze sobą połączyć i jak wykorzystywać poszczególne urządzenia wejścia i wyjścia. Posłużysz się w tym celu płytką Raspberry Pi i kompatybilnymi z nią podzespołami. Następnie napiszesz w Pythonie kod, dzięki któremu wzbogacisz swojego robota o sztuczną inteligencję i połączysz się z nim przez Wi-Fi za pomocą smartfona. Zdobędziesz również wiedzę, w jaki sposób realizować bardziej złożone projekty z zakresu robotyki, a także przygotujesz się, aby zwizualizować, zaprojektować, zbudować i zaprogramować robota według własnego pomysłu.

Z tą książką:

- skonfigurujesz Raspberry Pi pod kątem zbudowania robota ze sztuczną inteligencją
- podłączysz silniki i czujniki do Raspberry Pi
- zaprogramujesz inteligentnego robota
- wykorzystasz technologie rozpoznawania mowy i przetwarzania obrazu
- nauczysz się sterowania robotem ze sztuczną inteligencją przez Wi-Fi za pomocą smartfona
- zaczniesz samodzielnie projektować i budować roboty

Danny Staple — jest inżynierem programowania. Zawodowo programuje w Pythonie, zajmuje się też metodyką DevOps i automatyzacją. Zdobył duże doświadczenie w pracy z systemami wbudowanymi. Hobbystycznie buduje roboty i różne gadżety, jest też mentorem CoderDojo Ham, a jakiś czas temu prowadził kluby LEGO Robotics.

ISBN 978-83-283-8167-4
9 788328 381674
Cena: 99,00 zł

Packt