

Politechnika Warszawska

WYDZIAŁ MECHANICZNY  
ENERGETYKI I LOTNICTWA



Instytut Techniki Lotniczej i Mechaniki Stosowanej

# Praca dyplomowa inżynierska

na kierunku Automatyka i Robotyka  
w specjalności Robotyka

Implementacja algorytmu Bellmana- Forda do przeszukiwania labiryntu  
przez robota typu micromouse

**Marcin Baran**

259804

promotor  
prof. dr hab. inż. Teresa Zielińska

Warszawa, 2017



## Streszczenie

### Implementacja algorytmu Bellmana- Forda do przeszukiwania labiryntu przez robota typu Micromouse

Celem pracy było rozwiązanie problemu znajdowania najkrótszej drogi do środka labiryntu opisanego w regulaminie konkurencji Micromouse oraz implementacja wybranego algorytmu w robocie mobilnym. W dokumencie zawarto opis kolejnych etapów projektu, a także przedstawiono analizę jakości wdrożonego rozwiązania. Pierwszy rozdział poświęcony został przedstawieniu wymagań stawianym w konkurencji Micromouse (rozdział 1). Na ich podstawie powstała konstrukcja mechaniczna i elektroniczna robota mobilnego użytego w projekcie (rozdział 2). Następnym krokiem było wyprowadzenie modelu matematycznego kinematyki robota o napędzie różnicowym (rozdział 3), który stanowi podstawę przyjętej metody lokalizacji. Równania te wykorzystywane są do obliczania położenia na podstawie uprzednio wyliczonej pozycji oraz pomiarów z zastosowanych w konstrukcji czujników. W celu lokalizacji zastosowano algorytm SLAM wykorzystujący rozszerzony filtr Kalmana (rozdział 4). Filtr ten bardzo dobrze radzi sobie z estymacją procesów nieliniowych, a także daje możliwość zbierania parametrów pomiarowych z różnych czujników. Następnie, poprzez fuzję tych danych, uzyskiwana jest optymalna informacja o stanie procesu. Opisana metoda pozwala na wyznaczenie, w której części labiryntu znajduje się robot. W ten sposób możliwe jest utworzenie mapy, która wykorzystywana jest w rozwiązywaniu problemu najkrótszej ścieżki poprzez algorytm Bellmana-Forda (rozdział 5). W ostatnim etapie pracy przeprowadzono doświadczenia mające na celu sprawdzenie poprawności użytych metod. Na podstawie uzyskanych wyników możliwa była ocena jakości wykonanego projektu oraz wskazanie modyfikacji możliwych do wdrożenia w przyszłości (rozdział 6).

**Słowa kluczowe:** *Micromouse, robot mobilny, algorytm Bellmana-Forda, problem najkrótszej ścieżki, rozszerzony filtr Kalmana, lokalizacja*



## Abstract

### Implementation of Bellman-Ford algorithm for maze-solving by micromouse type robot

The aim of this thesis was solving the shortest path problem for a maze described in the Micromouse competition rules along with the implementation of a chosen algorithm on a mobile robot. The document presents detailed description of project with the quality assessment of applied methods. First chapter shows requirements for Micromouse competition (Chapter 1). Based on the rules, a mechanical and electronic platform of mobile robot was developed (Chapter 2). Next step was to obtain mathematical model of given differential-drive robot (Chapter 3) which was used in chosen approach. These equations are essential in determining the position based on previously known location and measurements acquired from sensors installed on the platform. In order to localize the robot, a SLAM algorithm using Extended Kalman Filter was implemented (Chapter 4). The afore mentioned filter is recommended for estimating non-linear processes and enables the use of sensor fusion. This method gives an optimal information about process state. It is possible to point out where in the maze the robot is. Created map of environment is used in Bellman-Ford algorithm for finding the shortest path through the maze (Chapter 5). The final stage of the project was to conduct experiments in order to assess if the methods used were correct. Based on obtained results the weaknesses and possible improvements were summarized (Chapter 6).

**Keywords:** *Micromouse, mobile robot, Bellman-Ford algorithm, shortest path problem, Extended Kalman Filter, localization*



## Oświadczenie autora (autorów) pracy

Świadom odpowiedzialności prawnej oświadczam, że przedstawiona praca dyplomowa:

- została napisana przeze mnie samodzielnie i nie zawiera treści uzyskanych w sposób niezgodny z obowiązującymi przepisami,
- nie była wcześniej przedmiotem procedur związanych z uzyskaniem tytułu zawodowego lub stopnia naukowego w wyższej uczelni.

Oświadczam ponadto, że niniejsza wersja pracy jest identyczna z załączoną wersją elektroniczną.

.....  
data

.....  
podpis autora (autorów) pracy

## Oświadczenie

Wyrażam zgodę / ~~nie wyrażam zgody~~<sup>\*1</sup> na udostępnianie osobom zainteresowanym mojej pracy dyplomowej. Praca może być udostępniana w pomieszczeniach biblioteki wydziałowej. Zgoda na udostępnienie pracy dyplomowej nie oznacza wyrażenia zgody na jej kopiowanie w całości lub w części.

Brak zgody nie oznacza ograniczenia dostępu do pracy dyplomowej osób:

- reprezentujących władze Politechniki Warszawskiej,
  - członków Komisji Akredytacyjnych,
  - funkcjonariuszy służb państwowych i innych osób uprawnionych, na mocy odpowiednich przepisów prawnych obowiązujących na terenie Rzeczypospolitej Polskiej, do swobodnego dostępu do materiałów chronionych międzynarodowymi przepisami o prawach autorskich.
- Brak zgody nie wyklucza także kontroli tekstu pracy dyplomowej w systemie antyplagiatowym.

.....  
data

.....  
podpis autora (autorów) pracy

\*1 - niepotrzebne skreślić





# Spis treści

<b>1</b>	<b>Wstęp</b>	<b>11</b>
1.1	Cel pracy . . . . .	11
1.2	Zastosowane narzędzia . . . . .	11
1.3	Zasady konkurencji Micromouse . . . . .	11
1.4	Przegląd stosowanych w konkurencji robotów . . . . .	12
1.4.1	Green Giant V4.2 . . . . .	12
1.4.2	Yukikaze . . . . .	13
<b>2</b>	<b>Opis użytej konstrukcji</b>	<b>15</b>
2.1	Opis mechaniki . . . . .	15
2.2	Opis elektroniki robota i czujników . . . . .	16
2.2.1	Żyroskop i akcelerometr - dane techniczne . . . . .	16
2.2.2	Enkodery - dane techniczne . . . . .	17
2.2.3	Czujniki odległości - dane techniczne . . . . .	17
2.2.4	Mikrokontroler - dane techniczne . . . . .	17
2.3	Przetwarzanie danych pomiarowych . . . . .	18
2.3.1	Pomiary z żyroskopu . . . . .	18
2.3.2	Pomiary z akcelerometru . . . . .	19
2.3.3	Pomiary z enkoderów . . . . .	20
2.4	Podsumowanie . . . . .	21
<b>3</b>	<b>Model kinematyki</b>	<b>22</b>
3.1	Przyjęte założenia . . . . .	22
3.2	Napęd różnicowy . . . . .	23
3.3	Układ odniesienia . . . . .	24
3.4	Wyprowadzenie wzorów analitycznych . . . . .	26
3.5	Przypadek szczególny . . . . .	29
3.6	Podsumowanie . . . . .	30
<b>4</b>	<b>Rozszerzony filtr Kalmana</b>	<b>31</b>
4.1	Charakterystyka filtru Kalmana . . . . .	31
4.2	Filtr Kalmana dla układów nieliniowych . . . . .	32
4.3	Algorytm rozszerzonego filtru Kalmana . . . . .	33
4.4	Implementacja filtru dla robota Micromouse . . . . .	34
4.5	Testy rozwiązania . . . . .	38
4.5.1	Testy orientacji . . . . .	38

4.5.2	Testy jazdy po prostej . . . . .	39
<b>5</b>	<b>Algorytm wyszukiwania drogi w labiryncie</b>	<b>40</b>
5.1	Sposoby przeszukiwania labiryntu . . . . .	42
5.1.1	Metoda podążania wzdłuż ściany . . . . .	42
5.1.2	Metoda wymuszania brute force . . . . .	42
5.1.3	Metody grafowe-algorytm Bellmana Forda . . . . .	43
5.2	Implementacja algorytmu Bellmana-Forda . . . . .	45
5.2.1	Mapa w pamięci robota . . . . .	45
5.2.2	Działanie algorytmu . . . . .	46
<b>6</b>	<b>Wnioski</b>	<b>51</b>
	<b>Bibliografia</b>	<b>53</b>
	<b>Spis rysunków</b>	<b>55</b>
	<b>Spis tablic</b>	<b>55</b>
<b>A</b>	<b>Załącznik: Schematy elektroniczne robota Ariadna</b>	<b>57</b>

# Rozdział 1

## Wstęp

### 1.1 Cel pracy

Jednym z częstych problemów pojawiających się w robotyce mobilnej jest odnajdowanie i planowanie drogi robota. Z tego względu na wielu organizowanych w Polsce, także za granicą zawodach robotycznych organizowana jest konkurencja Micromouse, której wymaganiem jest poprawne i jak najszybsze odnajdowanie drogi do celu. Celem tej pracy było opracowanie rozwiązania problemu mapowania nieznanego środowiska oraz implementacja algorytmu znajdowania najszybszej drogi do wskazanego punktu na potrzeby wyżej wspomnianej konkurencji oraz przygotowanego w ramach pracy przejściowej robota mobilnego Ariadna (rozdział 2).

### 1.2 Zastosowane narzędzia

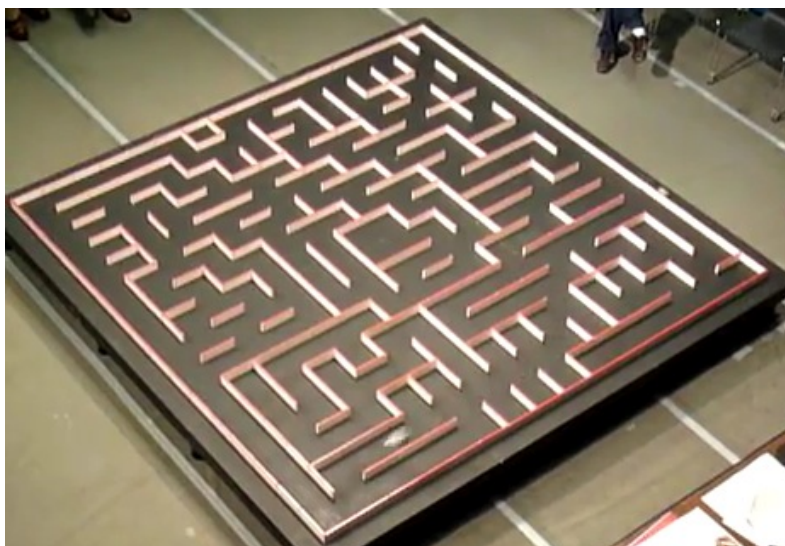
W ramach pracy napisano program w języku C służący do wykonania zadania przeszukiwania i przejazdu labiryntu przez robota mobilnego. Program został zaimplementowany na mikrokontrolerze STM32F103RBT6 firmy STMicrocontrollers używając do tego celu dedykowanej biblioteki HAL oraz środowiska Atollic True Studio oraz oprogramowania STM32CubeMX. Ponadto w celu przetestowania rozwiązania zastosowano oprogramowanie MATLAB do przetworzenia i przedstawienia w odpowiedniej formie danych uzyskanych podczas pracy robota.

### 1.3 Zasady konkurencji Micromouse

Zawody robotów typu Micromouse (źródła: [19, 13]) są jedną z najbardziej wymagających konkurencji ze względu na to, że jako jedne z niewielu wymagają od konstrukcji znajomości dokładnej, bieżącej pozycji. Dodatkowo oprogramowanie takiego robota często wiąże się z implementacją skomplikowanego algorytmu przeszukiwania.

Zadaniem jest przejazd od narożnika labiryntu do jego środka w jak najkrótszym czasie. Konstrukcje przy tym muszą być w pełni autonomiczne oraz zabroniona jest komunikacja podczas wykonywania zadania.

Labirynt w którym porusza się robot jest w kształcie kwadratu, podzielonego na pola o wymiarach 180 mm x 180 mm. Wszystkich takich pól jest 256 (16 rzędów i 16 kolumn) i są one dzielona od siebie ściankami o grubości 12 mm pomalowanymi białą farbą. W ten sposób wyznaczone są możliwe drogi przejazdu robota. Cel oznaczony jest jako 4 nie oddzielone pola znajdujące się w środku labiryntu.



Rysunek 1.1: Labirynt konkurencji Micromouse (źródło: [8])

Robot musi stworzyć w pamięci mapę nieznanego środowiska, rozpoczynając od jednego z narożników. Mapa ta jest niezbędna do wyznaczenia najszybszej trasy prowadzącej do środka. Później zadaniem konstrukcji jest jak najszybsze wykonanie przejazdu czasowego od narożnika do środka labiryntu po wyznaczonej trasie. Na wykonanie obu zadań przysługuje 10 minut.

Każde dotknięcie robota przez zawodnika skutkuje dodaniem kary do czasu wynikowego.

## 1.4 Przegląd stosowanych w konkurencji robotów

Zgodnie z założeniami zawodów konstrukcja robota ograniczona jest głównie przez wymiary komórek labiryntu. Stosowane technologie dotyczące wykrywania przeszkód w postaci ścianek, doboru napędu, czy metody lokalizacji robota nie są narzucone. Niemniej jednak zdecydowana większość konstrukcji opiera się o te same, sprawdzone rozwiązania.

### 1.4.1 Green Giant V4.2

Jest to robot amerykańskiego studenta-hobbysty zajmujący czołowe miejsca na zawodach Micromouse. Konstrukcję mechaniczną stanowi sztywna płytka PCB, na której zamontowane są 2 silniki. Sprawia to, że robot jest bardzo lekki. Robot posiada 4 koła, po 2 na każdą stronę, co zapewnia lepszą przyczepność, niż w konstrukcjach 2-kołowych oraz całkowicie niweluje konieczność używania ślizgaczy. Jednakże taki układ wymaga zastosowania specjalnej, zębatej przekładni mechanicznej w celu przełożenia napędu z silnika na oba koła. Ponadto robot 2-kołowy łatwiej wykonuje obroty wokół własnej osi (obroty w miejscu) niż robot 4-kołowy. (źródło: [21])



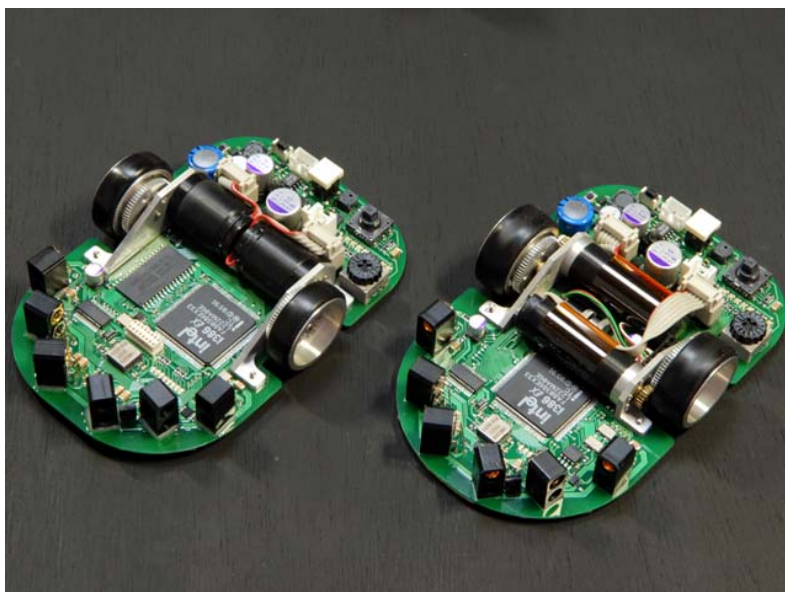
Rysunek 1.2: Robot Green Giant V4.2 (źródło: [21])

Tablica 1.1: Charakterystyka robota Green Giant V4.2(źródło: [21])

Szerokość	76mm
Długość	95mm
Wysokość	21.5mm
Waga	86g z baterią
Koła	Felgi z druku 3D; Opony gumowe Mini- z (9 mm)
Mikrokontroler	STM32F405RGT6
Silniki/Enkodery	Fauhalber 1717R(6V)/Enkoder magn. IE2 512
Przełożenie przekładni	16:60 M0.3
Bateria	120mah 2s1p
Czujniki odległości	TEFT4300+SFH4545(x4)
Żyroskop	ADXRS620

#### 1.4.2 Yukikaze

Japoński robot startujący w zawodach Micromouse zarówno w konkurencji labiryntu rozmiaru 16x16, jak i 32x32. Tak jak Green Giant, Yukikaze ma podwozie zrobione z płytki PCB z naniesionymi elementami elektronicznymi SMD. Jednakże różni się tym, że porusza się na 2 kołach, co sprawia, że jest zwrotniejszy, jednak musi opierać się z przodu i z tyłu na ślizgaczach. Zastosowanie napędu 2-kołowego zmniejsza dodatkowo masę całkowitą robota. W celu zminimalizowania szerokości robota, w Yukikaze zastosowano zębatą przekładnię mechaniczną, ułożyskowano wały kół w metalowych podporach oraz ustawiono silniki niewspółosiowo. Niskie zawieszenie powoduje zminimalizowanie różnicy nacisków na koła przy zakrętach, a tym samym ryzyko poślizgu. Ciekawostką jest fakt użycia jako czujników odległości diod IR z zarówno fotodiodami, jak i 2 układami odbiorników podczerwieni, które same w sobie filtrują odbierane impulsy świetlne. Użycie takiego zestawu podyktowane było zapewne sprawdzeniem różnicy odczytów na fotodiodach i odbiornikach. (źródło: [12])



Rysunek 1.3: Robot Yukikaze (źródło: [12])

Tablica 1.2: Charakterystyka robota Yukikaze (źródło: [12])

Szerokość	74mm
Długość	110mm
Wysokość	26mm
Waga	128g z baterią
Koła	Felgi z aluminium; Opony gumowe Mini-z
Mikrokontroler	Intel i80386EX 33.333MHz
Silniki/Enkodery	Fauhalber 1717R(6V)/ Enkodery magn. IE2 512
Przełożenie przekładni	16:60 M0.3
Bateria	Litowo- Polimerowa 250mAh 3S(11.1V)
Czujniki odległości	TPS708+SFH480-2(x6)+TAOS TSL262R(x2)
Żyroskop	Analog Devices ADXRS300

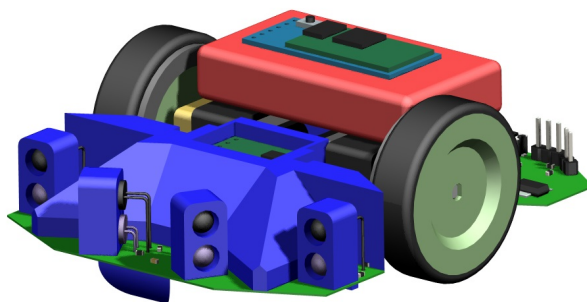
## Rozdział 2

# Opis użytej konstrukcji

Na potrzeby projektu w ramach pracy przejściowej autor pracy stworzył konstrukcję robota mobilnego opartą o sprawdzone rozwiązania stosowane dotychczas na zawodach. Poniżej znajduje się krótki opis mechaniczny platformy oraz charakterystyka zastosowanych czujników pomiarowych. Ze względu na wymagania stawiane w konkurencji, starano się zminimalizować długość i szerokość robota. Dobór odpowiednich czujników podyktowany był potrzebą lokalizacji robota w labiryncie, starano się także używać sprawdzonych, działających rozwiązań. Ze względu na małą masę wybrano silniki ze stosunkowo niewielkim przełożeniem, a dużą prędkością obrotową.

### 2.1 Opis mechaniki

Długość i szerokość robota odpowiednio wynoszą 100 mm x 80 mm. Podwozie stanowi samodzielnie wytrawiona płyta PCB z elementami elektronicznymi. Jej kształt został dobrany tak, aby koła nie wystawały poza obrys zewnętrzny co umożliwia uniknięcie zderzania kół o ściany labiryntu. Robot wyposażony został w napęd 2-kołowy (sterowanie różnicowe). Felgi mają średnicę 32 mm. Na nie zakładane są opony z poliuretanu typu Mini- Z o szerokości 9 mm. Obudowa przykręcana jest wraz z silnikami do płytki PCB. Wykonana została techniką druku 3D z materiału ABS. Dodatkowo do czujników odległości wydrukowano czarne obudowy w celu zabezpieczenia przed sygnałami zakłócającymi z zewnątrz. Jako napęd dobrano silniki Pololu HP z przekładnią 10:1, który osiąga  $3000 \text{ obr/min}$  i ma moment obrotowy  $0,3 \text{ kg}\cdot\text{cm}$  ( $0,029 \text{ Nm}$ ). Waga całkowita robota z baterią wynosi 130g.



Rysunek 2.1: Robot Ariadna wykorzystany w pracy-model CAD (źródło: materiały autora)

Tablica 2.1: Charakterystyka robota Ariadna (źródło: materiały autora)

Szerokość	80mm
Długość	100mm
Wysokość	40mm
Waga	130g z baterią
Koła	Felgi z druku 3D; Opony gumowe Mini- z (9 mm)
Mikrokontroler	STM32F106RGT6
Silniki/Enkodery	Pololu HP 10:1/Enkoder magn. AS5304A
Przełożenie przekładni	10:1
Bateria	120mah 2s1p
Czujniki odległości	LIRT3B-940+LIRED3B-940(x4)
Żyroskop	LSM6DS33

## 2.2 Opis elektroniki robota i czujników

### 2.2.1 Żyroskop i akcelerometr - dane techniczne

Wykorzystany został moduł MPU 6050 (źródło:[6]) posiadający akcelerometr i żyroskop LSM6DS33 oraz magnetometr LIS3MDL. Specyfikację modułu przedstawiono poniżej (opis ze strony dystrybutora):

- Rozmiar: 20 x 13 x 3 mm
- Napięcie zasilania: 2,5 V - 5,5 V
- Pobór prądu: 5 mA
- Wbudowany regulator napięcia
- Wbudowany konwerter napięć dla linii magistrali I2C (układ współpracuje zarówno z napięciami 5 V jak i 3,3 V)
- Pomiar wzdłuż 3 osi: X, Y, Z
- Interfejs komunikacyjny: I2C (TWI) Fast 400 kHz
- Rozdzielczość:
  - Akcelerometr: 16 bitów
  - Żyroskop: 16 bitów
  - Magnetometr: 16 bitów
- Zakresy pomiarowe (konfigurowalne):
  - Akcelerometr:  $\pm 2, \pm 4, \pm 8, \pm 16[G]$
  - Żyroskop:  $\pm 125, \pm 245, \pm 500, \pm 1000, \pm 2000[^\circ/s]$
  - Magnetometr:  $\pm 4, \pm 8, \pm 12, \pm 16[gauss]$



### 2.2.2 Enkodery - dane techniczne

Do realizacji projektu wybrano 2 magnetyczne enkodery inkrementalne (po jednym na każde koło) AS5304A (źródło: [2]) firmy AMS o specyfikacji:

- Napięcie zasilania 4.5- 5.5V;
- Pobór prądu 35 mA;
- Maksymalna odczytywana prędkość 20 m/s;
- Maksymalna rozdzielczość 4096 impulsów na obrót koła;
- Wyjście kwadraturowe AB

Do czujnika dobrano magnesy pierścieniowe AS5147P-HTMS montowane bezpośrednio na feldze w celu wyeliminowania wpływu luzów na przekładni silnika na pomiar. Nie eliminuje to jednak błędów związanych z poślizgiem koła.

### 2.2.3 Czujniki odległości - dane techniczne

Ze względu na duże rozmiary czujników odległości oferowanych w sprzedaży zdecydowano się zastosować jako czujnik odległości sprawdzone rozwiązanie, jakim jest para dioda IR i fototranzystor. Dioda wysyła wiązkę światła podczerwonego, którego moc maksymalna jest osiągana dla ściśle ustalonej wartości długości fali. Po odbiciu od przeszkody wiązka ta wraca i trafia do fototranzystora, którego parametry powinny być dobrane tak, aby reagować właśnie na daną długość fali. Sygnał z fototranzystora jest później odczytywany za pomocą przetwornika ADC. Im bliżej przeszkody jest robot, tym większa wartość zostanie odczytana. Problemem dla czujników mogą być inne sygnały świetlne zakłócające ich pomiar. Z tego względu dioda i fototranzystor zamknięte są w obudowie zawierającej filtr IR oraz zamocowane są na robocie w czarnych obudowach. Dobre czujniki (źródło: [10, 9]):

- 4 fototranzystory 940 nm LIRT3B-940 w 3 mm soczewce o zaciemnionej obudowie;
- 4 nadajniki IR 940 nm LIRED3B-940 w 3 mm soczewce, o kącie świecenia  $\pm 20^\circ$ .

### 2.2.4 Mikrokontroler - dane techniczne

Do przetwarzania danych napływających z czujników oraz obliczeń potrzebnych do mapowania labiryntu wykorzystany został mikrokontroler STM32F103RBT6 (źródło: [17]), którego specyfikację napisano poniżej:

- Rdzeń: 32-bitowy ARM Cortex M3
- Pamięć Flash: 128 kB
- Pamięć RAM: 20 kB
- Maksymalna częstotliwość taktowania rdzenia: 72 MHz
- Ilość liczników (timerów) 16-bitowych: 4
- Obsługa PWM

- Przetwornik ADC:
  - Rozdzielczość: 12 bitów
  - Ilość kanałów: 16 kanałów
- Przetwornik DAC:
  - Rozdzielczość: 12 bitów
  - Ilość kanałów: 2 kanałów
- Interfejsy: 2x SPI, 2x I2C, 3x USART, 1x CAN
- Napięcie zasilania: od 2 V do 3,6 V
- Obudowa: LQFP64 – 64 pin

## 2.3 Przetwarzanie danych pomiarowych

Dane uzyskiwane z czujników wymagają wstępnego przetworzenia, by można było ich użyć do pomiarów.

### 2.3.1 Pomiary z żyroskopu

Układ elektroniczny służy w tym zastosowaniu do pomiaru prędkości obrotowej robota w osi pionowej przechodzącej przez jego środek ciężkości. Umożliwia to określenie bieżącego kąta rotacji konstrukcji w obranym, globalnym układzie odniesienia. Żyroskop został ustawiony na zakres  $\pm 1000^\circ/s$ . Pomiary wykonywane są tylko w osi Z czujnika. Uzyskany wynik pomiaru pobierany jest z 2 rejestrów 8-bitowych i jest on przeliczany według wzoru:

$$\omega[^{\circ}/s] = \frac{\omega_{pomiar} \cdot zakres}{rozdzielczosc/2} \quad (2.1)$$

Gdzie:

- $\omega_{pomiar}$  jest wynikiem pobranym z żyroskopu
- $zakres$  wynosi  $1000^\circ/s$
- $rozdzielczosc$  wynosi  $2^{16}$

W powyższym równaniu rozdzielczość dzielona jest przez 2 ze względu na uwzględnienie znaku w wyniku pomiaru (uwzględniony jest zwrot prędkości kątowej).

Ponadto ze względu na występujący **dryf żyroskopu**, czyli błąd pomiarowy kumulujący się z czasem w urządzeniu, wyliczana jest średnia z 8000 pomiarów w czasie gdy robot znajduje się w bezruchu. Wyznaczona wartość dryfu jest następnie odejmowana przy wyznaczaniu wartości rzeczywistej odczytów.

Funkcja `checkDrift()` odpowiada za wyliczenie średniego błędu wynikającego z dryfu (makro `GyroRes` oznacza ustawioną rozdzielczość żyroskopu równą  $1000^\circ/s$ ). Poniżej podano fragment kodu:

---

```

void checkDriff() {
    for (int i = 0; i < 8000; ++i) {
        //Odczyt poprzez interfejs I2C danych pomiarowych z 2 rejestrów
        //8-bitowych żyroskopu:
        HAL_I2C_Mem_Read(&hi2c1, IMU_ADDRESS, 0x47, 1, &Data, 1, 1);
        HAL_I2C_Mem_Read(&hi2c1, IMU_ADDRESS, 0x48, 1, &DataZ, 1, 1);
        //Zapis wyniku pomiaru w postaci niemianowanej do zmiennej gyroDataZ:
        gyroDataZ = (Data << 8) | DataZ;
        //Obliczanie sumy pomiarów z 8000 kroków:
        rotZ += ((float) gyroDataZ * GyroRes) / 32767;
    }
    //Wyznaczanie wartości dryfu żyroskopu w stopniach/s i zapis do zmiennej driffZ:
    driffZ = rotZ / 8000.;
}

```

---

Uwzględniając poprawkę nowa wartość pomiaru wyliczana jest na podstawie wzoru:

$$\omega[^{\circ}/s] = \frac{\omega_{pomiar} \cdot zakres}{rozdzielczosc/2} - dryf \quad (2.2)$$

### 2.3.2 Pomiary z akcelerometru

Układ służy pomiarowi przyspieszenia w osiach leżących w płaszczyźnie ruchu robota: X (będącą osią kół robota) oraz prostopadłej do niej Y. Akcelerometr został ustawiony na zakres  $\pm 2g$ . Pomiary wykonywane są tylko wzdłuż osi X oraz Y urządzenia (z pominięciem osi Z). Uzyskany wynik pomiaru pobierany jest z 2 rejestrów 8-bitowych i jest on przeliczany według wzoru:

$$\begin{aligned}
 a_x\left[\frac{m}{s^2}\right] &= \frac{a_{xpomiar} \cdot zakres}{rozdzielczosc/2} \cdot 9.81\left[\frac{m}{s^2}\right] \\
 a_y\left[\frac{m}{s^2}\right] &= \frac{a_{ypomiar} \cdot zakres}{rozdzielczosc/2} \cdot 9.81\left[\frac{m}{s^2}\right]
 \end{aligned} \quad (2.3)$$

Gdzie:

- $a_{xpomiar}$  jest wynikiem pobranym z akcelerometru z pomiaru wzdłuż osi X
- $a_{ypomiar}$  jest wynikiem pobranym z akcelerometru z pomiaru wzdłuż osi Y
- $zakres$  wynosi 2 (w kontekście przyspieszenia ziemskiego)
- $rozdzielczosc$  wynosi  $2^{16}$

W powyższym równaniu rozdzielczość dzielona jest przez 2 ze względu na uwzględnienie znaku w wyniku pomiaru (uwzględniony jest zwrot prędkości kątowej).

Autor zaimplementował funkcję checkIMU(), która służy odczytaniu pomiarów z akcelerometru i żyroskopu. Poniżej napisany jest kod funkcji:

---

```

void checkIMU(void) {
    //Pomiary z akcelerometru:
    //Odczyt poprzez interfejs I2C danych pomiarowych akcelerometru (os X):
    HAL_I2C_Mem_Read(&hi2c1, IMU_ADDRESS, 0x3B, 1, &Data, 1, 1);

```

---

```

HAL_I2C_Mem_Read(&hi2c1, IMU_ADDRESS, 0x3C, 1, &DataZ, 1, 1);
//Zapis wyniku pomiaru w postaci niemianowanej do zmiennej accDataX:
accDataX = (Data << 8) | DataZ;
//Przeliczenie wartosci przyspieszenia na osi X:
accX = -((float) accDataX * 2.0) / 32767 * 9.81 - driffaccX;
//Odczyt poprzez interfejs I2C danych pomiarowych akcelerometru (os Y):
HAL_I2C_Mem_Read(&hi2c1, IMU_ADDRESS, 0x3D, 1, &Data, 1, 1);
HAL_I2C_Mem_Read(&hi2c1, IMU_ADDRESS, 0x3E, 1, &DataZ, 1, 1);
//Zapis wyniku pomiaru w postaci niemianowanej do zmiennej accDataY:
accDataY = -(Data << 8) | DataZ;
//Przeliczenie wartosci przyspieszenia na osi Y:
accY = ((float) accDataY * 2.0) / 32767 * 9.81 - driffaccY;
//Pomiary z zyroskopu:
//Odczyt poprzez interfejs I2C danych pomiarowych zyroskopu:
HAL_I2C_Mem_Read(&hi2c1, IMU_ADDRESS, 0x47, 1, &Data, 1, 1);
HAL_I2C_Mem_Read(&hi2c1, IMU_ADDRESS, 0x48, 1, &DataZ, 1, 1);
//Zapis wyniku pomiaru w postaci niemianowanej do zmiennej gyroDataZ:
gyroDataZ = (Data << 8) | DataZ;
//Przeliczenie wartosci predkosci katowej:
gyroZ = ((float) gyroDataZ * GyroRes) / 32767 - driffZ;
//Wylczenie wartosci kata obrotu poprzez proste calkowanie:
rotZ += gyroZ * (float) (HAL_GetTick() - lastSystemTick) / 1000;
}

```

Poza wyznaczeniem wartości chwilowych przyspieszeń oraz prędkości kątowej funkcja wylicza kąt obrotu poprzez całkowanie względem czasu odczytu z żyroskopu. Funkcja HAL GetTick() zwraca czas trwania programu mikrokontrolera od początku jego działania w *ms*, a zmienna lastSystemTick przechowuje wartość czasu trwania z poprzedniego kroku.

### 2.3.3 Pomiary z enkoderów

Układ ten służy głównie do wyznaczania prędkości kątowej kół robota. Przy obrocie enkoder zlicza ile impulsów zostało odebranych oraz zlicza je w celu podania względnego przemieszczenia kątowego. Przykładowo rozpoczynając pracę urządzenie wskazuje 0 impulsów. Przy pełnym obrocie koła zostanie doliczone 4096 impulsów (rozdzielczość enkodera). Dalszy obrót spowoduje wyzerowanie licznika i rozpoczęcie liczenia od 0. Przy zmianie kierunku liczba impulsów maleje od 4096 do 0. Ze względu na występujący przeskok przy przekroczeniu zakresu należy sprawdzać kiedy następuje wyzerowanie lub przejście do wartości 4096 oraz odpowiednio korygować odczyty. Prędkość kątowa kół mierzona jest w sposób różnicowy. Można ją obliczyć za pomocą wzoru (2.4):

$$\omega = \frac{\Delta N \cdot 2\pi}{\Delta t} \quad (2.4)$$

Gdzie:

- $\Delta N$  to zmiana ilości zliczonych impulsów
- $\Delta t$  to okres

Autor zaimplementował funkcję checkVelocity(), która służy odczytaniu i przetworzeniu pomiarów z enkoderów. Poniżej napisany jest kod funkcji:

---

```

void checkVelocity(void) {
    //Odczytanie wartosci licznikow lewego i prawego enkodera:
    EncoderLeft = TIM1->CNT;
    EncoderRight = TIM2->CNT;
    //Sprawdzenie warunkow przekroczenia zakresu dla lewego enkodera:
    if (EncoderLeft >= 0 && EncoderLeft <= 1096 && PreviousEncoderLeft >= 3000
        && PreviousEncoderLeft <= 4096) {
        AngularVelocityLeft = EncoderLeft - PreviousEncoderLeft + Ticks;
    } else if (EncoderLeft <= 4096 && EncoderLeft >= 3000
        && PreviousEncoderLeft >= 0 && PreviousEncoderLeft <= 1096) {
        AngularVelocityLeft = EncoderLeft - PreviousEncoderLeft - Ticks;
    } else {
        AngularVelocityLeft = EncoderLeft - PreviousEncoderLeft;
    }
    //Sprawdzenie warunkow przekroczenia zakresu dla prawego enkodera:
    if (EncoderRight >= 0 && EncoderRight <= 1096
        && PreviousEncoderRight >= 3000 && PreviousEncoderRight <= 4096) {
        AngularVelocityRight = EncoderRight - PreviousEncoderRight + Ticks;
    } else if (EncoderRight <= 4096 && EncoderRight >= 3000
        && PreviousEncoderRight >= 0 && PreviousEncoderRight <= 1096) {
        AngularVelocityRight = EncoderRight - PreviousEncoderRight - Ticks;
    } else {
        AngularVelocityRight = EncoderRight - PreviousEncoderRight;
    }
    //Ustawienie wartosci poprzednich odczytow na biezace:
    PreviousEncoderRight = EncoderRight;
    PreviousEncoderLeft = EncoderLeft;
    //Przeliczenie odczytow roznicowych enkoderow na predkosci katowe [rad/s]:
    AngularVelocityLeft = AngularVelocityLeft / 4096 * 2 * PI
        / (HAL_GetTick() - lastSystemTick) * 1000;
    AngularVelocityRight = AngularVelocityRight / 4096 * 2 * PI
        / (HAL_GetTick() - lastSystemTick) * 1000;
}

```

---

W celu wykonania pomiaru różnicowego zapamiętywane są wartości ilości impulsów z poprzedniej chwili (*PreviousEncoderRight* oraz *PreviousEncoderLeft*), które odejmowane są od wartości z chwili obecnej (*EncoderRight* oraz *EncoderLeft*). Zmienna *lastSystemTick* przechowuje wartość czasu trwania z poprzedniego kroku. Wartości bieżące prędkości kątowych kół zapisywane są do zmiennych *AngularVelocityLeft* i *AngularVelocityRight*.

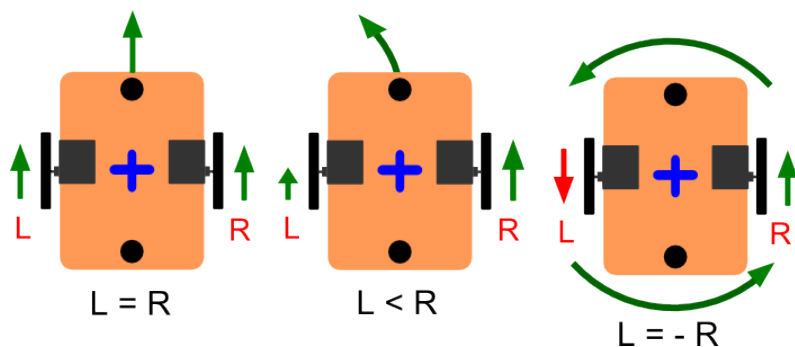
## 2.4 Podsumowanie

W celu opracowania modelu konstrukcji mechanicznej robota wykorzystano oprogramowanie Solidworks 2015. Obudowa wydrukowana została w technologii druku 3D z pomocą programu Simplify 3D. Płytkę elektroniczną samodzielnie zaprojektowano oraz wytrawiono korzystając z pakietu Altium Designer 2015. Schematy elektroniczne dołączone zostały w załączniku (załącznik A). Konstrukcja robota wykonana została w ramach pracy przejściowej autora.

## Rozdział 3

# Model kinematyki

Wykorzystywany w pracy robot jest konstrukcją o napędzie różnicowym (źródła: [14, 4, 7]). Oznacza to, że do sterowania platformą wykorzystywane są dwa odrębnie napędzane koła znajdujące się w tej samej osi. Robot może więc zmieniać kierunek jazdy poprzez zmianę prędkości obrotowej kół, jak na rysunku 3.1.



Rysunek 3.1: Schemat sterowania robotem dwukołowym o napędzie różnicowym ( od lewej: jazda na wprost, skręt po łuku w lewo, skręt w miejscu w lewo) (źródło: [15])

### 3.1 Przyjęte założenia

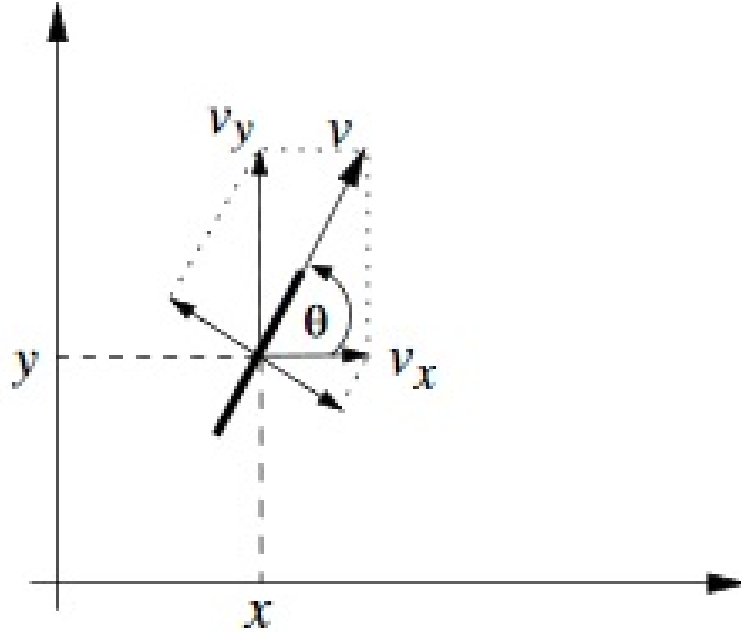
W celu uproszczenia i umożliwienia wyprowadzenia równań ruchu konieczne było przyjęcie pewnych założeń wynikających zarówno z rodzaju wykorzystywanego napędu, jak i środowiska, w którym poruszać się będzie robot. Z tego względu podstawą do wyprowadzenia równań kinematyki są następujące założenia:

- robot porusza się po płaszczyźnie idealnie poziomej
- koła mają punktowy kontakt z podłożem
- koła nie odkształcają się w trakcie ruchu
- brak możliwości sterowania ruchu kół w kierunku poprzecznym
- osie skrętu są prostopadłe do powierzchni
- koła osadzone są na sztywnym wale

- przyjęty model kół standardowych nieskrętnych
- przyspieszenia liniowe i prędkość kątowna są stałe w czasie

## 3.2 Napęd różnicowy

Podstawą wyprowadzenia modelu kinematyki dla napędu różnicowego są więzy (źródło: [14]) wynikające z braku możliwości ruchu oraz poślizgu w kierunku poprzecznym kół.



Rysunek 3.2: Więzy kinematyczne napędu różnicowego (źródło: [7])

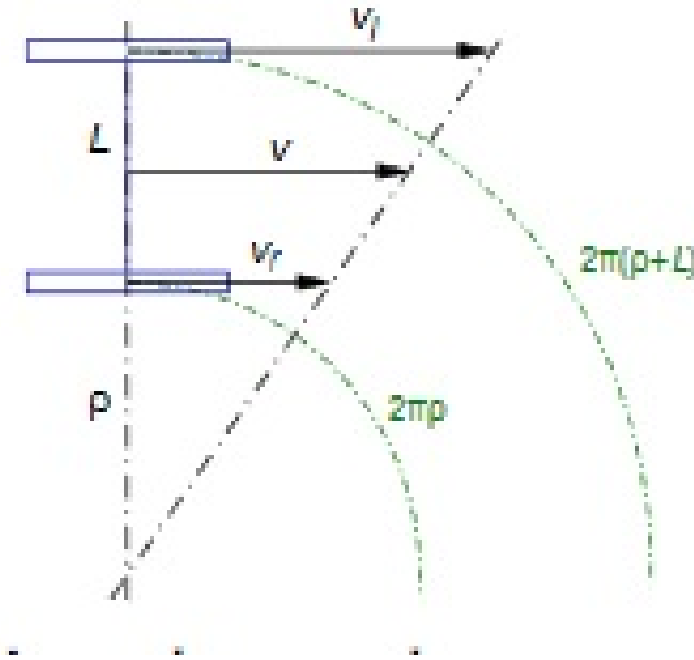
Oznacza to, że w każdej chwili ruchu wypadkowy wektor prędkości w kierunku poprzecznym do kierunku koła musi być zrównoważony:

$$v_x \cdot \sin(\theta) - v_y \cdot \cos(\theta) = 0 \quad (3.1)$$

Ponadto oba koła muszą spełniać powyższą zależność. Z tego względu możliwe jest sprowadzenie układu różnicowego do modelu monocykla. Opisany jest on za pomocą prędkości liniowej wynikającej z ruchu obrotowego koła napędowego oraz prędkości obrotowej ( $\omega$ ) względem osi prostopadłej do powierzchni i przechodzącej przez środek osi łączącej koła napędowe. Stąd otrzymane zostają równania postaci:

$$\begin{aligned} v_x &= v \cdot \cos(\theta) \\ v_y &= v \cdot \sin(\theta) \\ \dot{\theta} &= \omega \end{aligned} \quad (3.2)$$

Znając prędkość obrotową obu kół możemy wyznaczyć zarówno prędkość wypadkową  $v$  oraz prędkość obrotową  $\omega$  w celu sprowadzenia do układu monocykla opisanego równaniami (3.2).



Rysunek 3.3: Sprowadzanie układu różnicowego do modelu monocykla (źródło: [7])

W przypadku jazdy robota po łuku zmiana kierunku ruchu następuje poprzez obrót wokół osi prostopadłej do powierzchni, oddalonej od środka osi kół o  $L/2 + \rho$ . W danej chwili można ruch ten przedstawić jako ruch po okręgu o promieniu  $L/2 + \rho$  z prędkością  $v$ . Stąd liniową prędkość wypadkową  $v$  oraz prędkość obrotową  $\omega$  można wyznaczyć za pomocą wzorów:

$$v = \frac{v_l + v_r}{2} \quad (3.3)$$

$$\omega = \frac{v_r - v_l}{L} \quad (3.4)$$

Znając ponadto średnicę kół ( $D$ ) można prędkość wypadkową oraz prędkość obrotową przedstawić w postaci:

$$v = \frac{(\omega_l + \omega_r) \cdot D}{2} \quad (3.5)$$

$$\omega = \frac{(\omega_r - \omega_l) \cdot D}{L} \quad (3.6)$$

Gdzie  $\omega_l$  i  $\omega_r$  to prędkości obrotowe odpowiednio lewego i prawego koła. Wartość średnicy kół  $D$  oraz rozstaw kół  $L$  wynoszą odpowiednio:

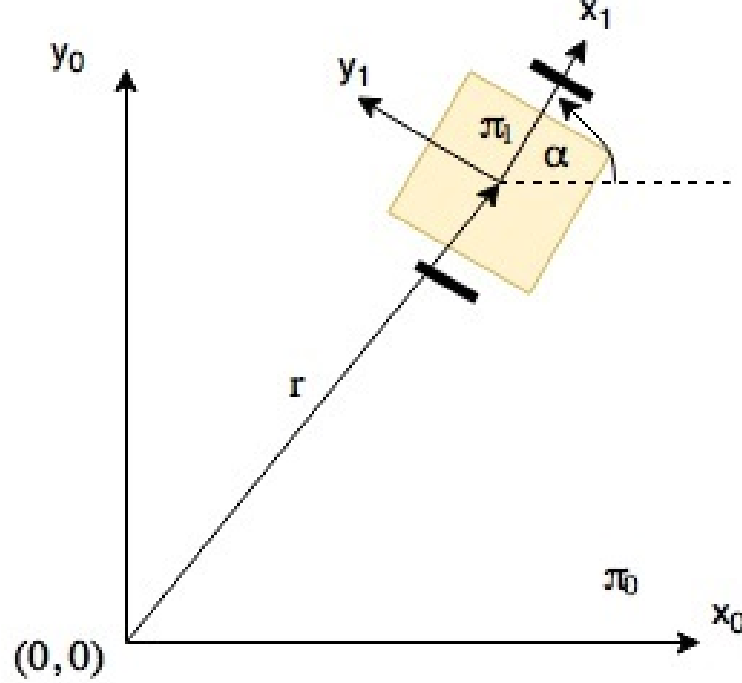
$$\begin{aligned} L &= 80mm \\ D &= 36mm \end{aligned} \quad (3.7)$$

### 3.3 Układ odniesienia

Ze względu na specyfikę środowiska, w którym operuje robot, najlepszym rozwiązaniem jest wyznaczanie pozycji robota w nieruchomym układzie odniesienia związanym z punktem początkowym labiryntu ( $\pi_0$ ). Znając współrzędne robota w takim układzie łatwo można określić, w



której komórce aktualnie znajduje się konstrukcja- zagadnienie szerzej omówione zostaje w rozdziale (5). Parametry ruchu takie jak prędkość wypadkowa  $v$  mierzone są w układzie odniesienia ( $\pi_1$ ) związanym z robotem. Ponadto zwrot osi układu przyjęty został w ten sposób, że oś  $y$  skierowana jest prostopadłe do osi kół i zwrócona w przód robota.



Rysunek 3.4: Pozycja robota w obranym układzie odniesienia (źródło: materiały autora)

Transformacja wektora prędkości  $\mathbf{v}$  z układu  $\pi_1$  do  $\pi_0$  polega na wymnożeniu wektora przez macierz rotacji (źródło: [14]). Ze względu na przyjęte założenia (brak możliwości ruchu w kierunku poprzecznym) składowa w kierunku osi  $x_1$  wektora  $\mathbf{v}$  wynosi w każdej chwili 0. Stąd:

$$\mathbf{v}^0 = \mathbf{R}\mathbf{v}^1 \quad (3.8)$$

Gdzie:

$$\mathbf{R} = \begin{bmatrix} \cos(\alpha(t)) & -\sin(\alpha(t)) \\ \sin(\alpha(t)) & \cos(\alpha(t)) \end{bmatrix} \quad (3.9)$$

jest macierzą rotacji z układu  $\pi_1$  do  $\pi_0$ . Stąd wzór na prędkość w układzie  $\pi_0$  wynosi:

$$\mathbf{v}^0 = \begin{bmatrix} \cos(\alpha(t)) & -\sin(\alpha(t)) \\ \sin(\alpha(t)) & \cos(\alpha(t)) \end{bmatrix} \begin{bmatrix} 0 \\ v \end{bmatrix} = \begin{bmatrix} -v \sin(\alpha(t)) \\ v \cos(\alpha(t)) \end{bmatrix} \quad (3.10)$$

Analogicznie rozumowanie przebiega dla wektora przyspieszeń  $\mathbf{a}$ , przy czym w tym wypadku rozsądnym jest założenie, że składowa  $a_x$  wektora  $\mathbf{a}^0$  może być niezerowa w skutek występującego na zakręcie poślizgu kół w kierunku poprzecznym. Ze względu na pomiar przyspieszenia

przez akcelerometr zarówno w osi  $x_1$ , jak i  $y_1$  (pomiar prędkości odbywa się tylko wzdłuż kierunku  $y_1$ ) oraz wykorzystaniu rozszerzonego filtru Kalmana możliwe staje się wykrycie tego typu niedoskonałości konstrukcji:

$$\mathbf{a}^0 = \begin{bmatrix} \cos(\alpha(t)) & -\sin(\alpha(t)) \\ \sin(\alpha(t)) & \cos(\alpha(t)) \end{bmatrix} \begin{bmatrix} a_x \\ a_y \end{bmatrix} = \begin{bmatrix} a_x \cos(\alpha(t)) - a_y \sin(\alpha(t)) \\ a_x \sin(\alpha(t)) + a_y \cos(\alpha(t)) \end{bmatrix} \quad (3.11)$$

### 3.4 Wyprowadzenie wzorów analitycznych

Wykorzystując wyprowadzone równanie na wektor przyspieszeń (3.11), poprzez rozwiązanie zagadnienia prostego kinematyki, można wyznaczyć wektor położenia  $\mathbf{r}(t)$ .

Przy wyprowadzaniu równań założono, że ruch konstrukcji jest złożeniem ruchu w równych odstępach czasu  $\delta t$ . Daje to możliwość zmniejszenia wpływu błędów wynikających z błędów przyjętego modelu (im mniejsze  $\Delta t$ , tym mniejszy wpływ niedoskonałości modelu). Aktualny wektor położenia robota (reprezentowany przez współrzędne  $x, y$  oraz kąt obrotu  $\alpha$  w układzie  $\pi_0$ , w czasie  $t$  można wyznaczyć na podstawie znajomości wektora z poprzedniej chwili  $\mathbf{r}(t - \Delta t)$  oraz jego pochodnej względem czasu ( $\dot{\mathbf{r}}(t - \Delta t)$ ). Znana jest druga pochodna  $\ddot{\mathbf{r}}(t)$  (wyrażenie na przyspieszenie (3.11)) postaci:

$$\ddot{\mathbf{r}}(t) = \begin{bmatrix} \mathbf{a}^0(t) \\ 0 \end{bmatrix} = \begin{bmatrix} a_x \cos(\alpha(t)) - a_y \sin(\alpha(t)) \\ a_x \sin(\alpha(t)) + a_y \cos(\alpha(t)) \\ 0 \end{bmatrix} \quad (3.12)$$

oraz następujące warunki początkowe:

- wektor prędkości z poprzedniej chwili ( $t - \Delta t$ ):

$$\dot{\mathbf{r}}(t - \Delta t) = \begin{bmatrix} \dot{x}(t - \Delta t) \\ \dot{y}(t - \Delta t) \\ \dot{\alpha}(t - \Delta t) \end{bmatrix} = \begin{bmatrix} \dot{x}_p \\ \dot{y}_p \\ \omega_p \end{bmatrix} \quad (3.13)$$

- wektor położenia z poprzedniej chwili ( $t - \Delta t$ ):

$$\mathbf{r}(t - \Delta t) = \begin{bmatrix} x(t - \Delta t) \\ y(t - \Delta t) \\ \alpha(t - \Delta t) \end{bmatrix} = \begin{bmatrix} x_p \\ y_p \\ \alpha_p \end{bmatrix} \quad (3.14)$$

Po jednokrotnym scałkowaniu równań (3.12) względem czasu otrzymane zostaje wyrażenie na wektor prędkości  $\dot{\mathbf{r}}(t)$ :

$$\dot{\mathbf{r}}(t) = \begin{bmatrix} \dot{x}(t) \\ \dot{y}(t) \\ \dot{\alpha}(t) \end{bmatrix} = \int_0^{\Delta t} \begin{bmatrix} a_x \cos(\alpha(t)) - a_y \sin(\alpha(t)) \\ a_x \sin(\alpha(t)) + a_y \cos(\alpha(t)) \\ 0 \end{bmatrix} dt \quad (3.15)$$

Ze względu na to, że prędkość  $\dot{x}(t)$  oraz  $\dot{y}(t)$  zależą od aktualnej orientacji robota  $\alpha(t)$  najpierw należy wyprowadzić zależność kąta obrotu od czasu:

$$\dot{\alpha}(t) = 0 \cdot t + C = \omega_p \quad (3.16)$$

Końcowa wartość wynika bezpośrednio z warunków początkowych (3.13) oraz przyjętego założenia modelu mówiącego że prędkość kątowa jest stała w czasie. Wyrażenie na położenie kątowe  $\alpha(t)$  otrzymane jest poprzez scałkowanie wyrażenia prędkości kątowej (3.16) względem czasu:

$$\alpha(t) = \int_0^{\Delta t} \omega_p dt = \omega_p \cdot t + C = \omega_p \cdot \Delta t + \alpha_p \quad (3.17)$$

Wartość  $\alpha_p$  wynika z warunków początkowych (3.14). Znając funkcję orientacji od czasu można rozwiązać równania różniczkowe położenia liniowych robota. W celu przejrzystości całkowania pośrednie zapisano oddzielnie:

$$\int \sin(\alpha_p + \omega_p \cdot t) dt = -\frac{1}{\omega_p} \cdot \cos(\alpha_p + \omega_p \cdot t) + C \quad (3.18)$$

$$\int \cos(\alpha_p + \omega_p \cdot t) dt = \frac{1}{\omega_p} \cdot \sin(\alpha_p + \omega_p \cdot t) + C \quad (3.19)$$

Wyprowadzone wyrażenia (3.18) i (3.19) użyto w poniższych równaniach:

$$\begin{aligned} \dot{x}(t) &= \int_0^{\Delta t} (a_x \cos(\alpha_p + \omega_p \cdot t) - a_y \sin(\alpha_p + \omega_p \cdot t)) dt = \\ &= \left[ \frac{1}{\omega_p} \cdot (a_x \sin(\alpha_p + \omega_p \cdot t) + a_y \cos(\alpha_p + \omega_p \cdot t)) \right]_0^{\Delta t} + C = \\ &= \frac{1}{\omega_p} \cdot (a_x \sin(\alpha_p + \omega_p \cdot \Delta t) - a_x \sin(\alpha_p) + a_y \cos(\alpha_p + \omega_p \cdot \Delta t) - a_y \cos(\alpha_p)) + C \end{aligned} \quad (3.20)$$

$$\begin{aligned} \dot{y}(t) &= \int_0^{\Delta t} (a_x \sin(\alpha_p + \omega_p \cdot t) + a_y \cos(\alpha_p + \omega_p \cdot t)) dt = \\ &= \left[ \frac{1}{\omega_p} \cdot (-a_x \cos(\alpha_p + \omega_p \cdot t) + a_y \sin(\alpha_p + \omega_p \cdot t)) \right]_0^{\Delta t} + C = \\ &= \frac{1}{\omega_p} \cdot (-a_x \cos(\alpha_p + \omega_p \cdot \Delta t) + a_x \cos(\alpha_p) + a_y \sin(\alpha_p + \omega_p \cdot \Delta t) - a_y \sin(\alpha_p)) + C \end{aligned}$$

Stąd ze względu na warunki początkowe (3.13) końcowe wyrażenia na prędkości liniowe przedstawiają się następująco:

$$\begin{aligned} \dot{x}(t) &= \frac{1}{\omega_p} \cdot (a_x \sin(\alpha_p + \omega_p \cdot \Delta t) - a_x \sin(\alpha_p) + a_y \cos(\alpha_p + \omega_p \cdot \Delta t) - a_y \cos(\alpha_p)) + \dot{x}_p \\ \dot{y}(t) &= \frac{1}{\omega_p} \cdot (-a_x \cos(\alpha_p + \omega_p \cdot \Delta t) + a_x \cos(\alpha_p) + a_y \sin(\alpha_p + \omega_p \cdot \Delta t) - a_y \sin(\alpha_p)) + \dot{y}_p \end{aligned} \quad (3.21)$$

W tym momencie uzyskany został wektor prędkości w czasie. Wyrażony jest za pomocą równań (3.21) i (3.16):

$$\dot{\mathbf{r}}(t) = \begin{bmatrix} \frac{1}{\omega_p} \cdot (a_x \sin(\alpha_p + \omega_p \cdot \Delta t) - a_x \sin(\alpha_p) + a_y \cos(\alpha_p + \omega_p \cdot \Delta t) - a_y \cos(\alpha_p)) + \dot{x}_p \\ \frac{1}{\omega_p} \cdot (-a_x \cos(\alpha_p + \omega_p \cdot \Delta t) + a_x \cos(\alpha_p) + a_y \sin(\alpha_p + \omega_p \cdot \Delta t) - a_y \sin(\alpha_p)) + \dot{y}_p \\ \omega_p \end{bmatrix} \quad (3.22)$$

Znając wektor prędkości oraz warunki początkowe (3.14), po scałkowaniu względem czasu, otrzy-

many zostaje ostatecznie wektor położenia  $\mathbf{r}(t)$  od czasu. W poniższych obliczeniach korzystano także z tożsamości (3.18) i (3.19):

$$\begin{aligned}
x(t) &= \int_0^{\Delta t} \left( \frac{1}{\omega_p} \cdot (a_x \sin(\alpha_p + \omega_p \cdot t) - a_x \sin(\alpha_p) + a_y \cos(\alpha_p + \omega_p \cdot t) - a_y \cos(\alpha_p)) + \dot{x}_p \right) dt = \\
&= \left[ \frac{1}{\omega_p^2} \cdot (-a_x \cos(\alpha_p + \omega_p \cdot \Delta t) + a_x \cos(\alpha_p) + a_y \sin(\alpha_p + \omega_p \cdot \Delta t) - a_y \sin(\alpha_p)) + \right. \\
&\quad \left. + \left( \dot{x}_p + \frac{1}{\omega_p} \cdot (-a_x \sin(\alpha_p) + a_y \cos(\alpha_p)) \cdot t \right) \right]_0^{\Delta t} + C = \\
&= \frac{1}{\omega_p^2} \cdot (-a_x \cos(\alpha_p + \omega_p \cdot \Delta t) + a_x \cos(\alpha_p) + a_y \sin(\alpha_p + \omega_p \cdot \Delta t) - a_y \sin(\alpha_p)) + \\
&\quad + \left( \dot{x}_p + \frac{1}{\omega_p} \cdot (-a_x \sin(\alpha_p) + a_y \cos(\alpha_p)) \cdot \Delta t + x_p \right) \\
y(t) &= \int_0^{\Delta t} \left( \frac{1}{\omega_p} \cdot (-a_x \cos(\alpha_p + \omega_p \cdot t) + a_x \cos(\alpha_p) + a_y \sin(\alpha_p + \omega_p \cdot t) - a_y \sin(\alpha_p)) + \dot{y}_p \right) dt = \\
&= \left[ \frac{1}{\omega_p^2} \cdot (-a_x \sin(\alpha_p + \omega_p \cdot t) - a_y \cos(\alpha_p + \omega_p \cdot t)) + \right. \\
&\quad \left. + \left( \dot{y}_p + \frac{1}{\omega_p} \cdot (a_x \cos(\alpha_p) - a_y \sin(\alpha_p)) \cdot t \right) \right]_0^{\Delta t} + C = \\
&= \frac{1}{\omega_p^2} \cdot (-a_x \sin(\alpha_p + \omega_p \cdot \Delta t) + a_x \sin(\alpha_p) - a_y \cos(\alpha_p + \omega_p \cdot \Delta t) + a_y \cos(\alpha_p)) + \\
&\quad + \left( \dot{y}_p + \frac{1}{\omega_p} \cdot (a_x \cos(\alpha_p) - a_y \sin(\alpha_p)) \cdot \Delta t + y_p \right)
\end{aligned} \tag{3.23}$$

Wektor położenia opisywany jest więc przez równania (3.23) oraz (3.17) i przedstawia się następująco:

$$\begin{aligned}
\mathbf{r}(t) &= \begin{bmatrix} \frac{1}{\omega_p^2} \cdot (-a_x \cos(\alpha_p + \omega_p \cdot \Delta t) + a_x \cos(\alpha_p) + a_y \sin(\alpha_p + \omega_p \cdot \Delta t) - a_y \cos(\alpha_p)) \\ \frac{1}{\omega_p^2} \cdot (-a_x \sin(\alpha_p + \omega_p \cdot \Delta t) + a_x \sin(\alpha_p) - a_y \cos(\alpha_p + \omega_p \cdot \Delta t) + a_y \cos(\alpha_p)) \\ \omega_p \cdot \Delta t \end{bmatrix} \\
&\quad + \begin{bmatrix} \left( \dot{x}_p + \frac{1}{\omega_p} \cdot (-a_x \sin(\alpha_p) + a_y \cos(\alpha_p)) \cdot \Delta t \right) \\ \left( \dot{y}_p + \frac{1}{\omega_p} \cdot (a_x \cos(\alpha_p) - a_y \sin(\alpha_p)) \cdot \Delta t \right) \\ 0 \end{bmatrix} + \begin{bmatrix} x_p \\ y_p \\ \alpha_p \end{bmatrix}
\end{aligned} \tag{3.24}$$

Uzyskany wynik przedstawia wektor położenia w obranym układzie współrzędnych  $\pi_0$  w zależności od poprzedniego wektora położenia  $\mathbf{r}(t - \Delta t)$  oraz ilości czasu mijającego od poprzedniej aktualizacji wektora  $\mathbf{r}(t)$  równej  $\Delta t$ .

### 3.5 Przypadek szczególny

Równanie (3.24) nie opisuje całkowicie ruchu robota ze względu na to, że nie jest określone dla wartości  $\omega_p = 0$ . Jest to szczególny przypadek, który rozpatrzony jest poniżej osobno. Wychodząc od równań (3.11), które w tym przypadku zostają uproszczone oraz zakładając te same warunki początkowe (3.13),(3.14) można analogicznie wyprowadzić zależności na wektor położenia:

$$\omega_p = 0 \Rightarrow \ddot{\mathbf{r}}(t) = \begin{bmatrix} a_x \cos(\alpha_p) - a_y \sin(\alpha_p) \\ a_x \sin(\alpha_p) + a_y \cos(\alpha_p) \\ 0 \end{bmatrix} \quad (3.25)$$

W tym przypadku zależność orientacja robota jest stała w czasie, ze względu na to, że prędkość obrotowa wynosi 0.

$$\omega(t) = \omega_p = 0 \Rightarrow \alpha(t) = 0 \quad (3.26)$$

Poprzez jednokrotne scałkowanie równań (3.25) otrzymujemy wektor prędkości:

$$\begin{aligned} \dot{x}(t) &= \int_0^{\Delta t} a_x \cos(\alpha_p) - a_y \sin(\alpha_p) dt = [(a_x \cos(\alpha_p) - a_y \sin(\alpha_p)) \cdot t]_0^{\Delta t} + C = \\ & (a_x \cos(\alpha_p) - a_y \sin(\alpha_p)) \cdot \Delta t + \dot{x}_p \\ \dot{y}(t) &= \int_0^{\Delta t} a_x \sin(\alpha_p) + a_y \cos(\alpha_p) dt = [(a_x \sin(\alpha_p) + a_y \cos(\alpha_p)) \cdot t]_0^{\Delta t} + C = \\ & (a_x \sin(\alpha_p) + a_y \cos(\alpha_p)) \cdot \Delta t + \dot{y}_p \\ \dot{\alpha}(t) &= 0 \end{aligned} \quad (3.27)$$

$$\dot{\mathbf{r}}(t) = \begin{bmatrix} (a_x \cos(\alpha_p) - a_y \sin(\alpha_p)) \cdot \Delta t + \dot{x}_p \\ (a_x \sin(\alpha_p) + a_y \cos(\alpha_p)) \cdot \Delta t + \dot{y}_p \\ 0 \end{bmatrix} \quad (3.28)$$

Znając wektor prędkości wyznaczony zostaje wektor położenia  $\mathbf{r}(t)$ :

$$\begin{aligned} x(t) &= \int_0^{\Delta t} (a_x \cos(\alpha_p) - a_y \sin(\alpha_p)) \cdot t + \dot{x}_p dt = [(a_x \cos(\alpha_p) - a_y \sin(\alpha_p)) \cdot \frac{t^2}{2} + \dot{x}_p \cdot t]_0^{\Delta t} + C = \\ & (a_x \cos(\alpha_p) - a_y \sin(\alpha_p)) \cdot \frac{\Delta t^2}{2} + \dot{x}_p \cdot \Delta t + x_p \\ y(t) &= \int_0^{\Delta t} (a_x \sin(\alpha_p) + a_y \cos(\alpha_p)) \cdot t + \dot{y}_p dt = [(a_x \sin(\alpha_p) + a_y \cos(\alpha_p)) \cdot \frac{t^2}{2} + \dot{y}_p \cdot t]_0^{\Delta t} + C = \\ & (a_x \sin(\alpha_p) + a_y \cos(\alpha_p)) \cdot \frac{\Delta t^2}{2} + \dot{y}_p \cdot t + y_p \\ \alpha(t) &= \alpha_p \end{aligned} \quad (3.29)$$

$$\mathbf{r}(t) = \begin{bmatrix} (a_x \cos(\alpha_p) - a_y \sin(\alpha_p)) \cdot \frac{\Delta t^2}{2} + \dot{x}_p \cdot \Delta t + x_p \\ (a_x \sin(\alpha_p) + a_y \cos(\alpha_p)) \cdot \frac{\Delta t^2}{2} + \dot{y}_p \cdot t + y_p \\ \alpha_p \end{bmatrix} \quad (3.30)$$

### 3.6 Podsumowanie

Zależności (3.30) oraz (3.24) stanowią pełne równania modelu kinematyki robota mobilnego o napędzie różnicowym w przyjętym, płaskim układzie współrzędnych ((3.30)- w przypadku, gdy  $\omega(t) = 0$ ). Znając je, a także odczytując pomiary z enkoderów, akcelometru i żyroskopu, możliwe staje się wyznaczenie pozycji robota w przestrzeni w kolejnych chwilach. Przyspieszenia liniowe  $a_x, a_y$  oraz prędkość kątowna  $\omega_p$  są znane z odczytów akcelometru i żyroskopu. Enkodery dają możliwość wyznaczenia prędkości obrotowej poszczególnych kół. Do obliczeń potrzebne są ponadto odpowiednie przekształcenia pomiarów z czujników na wartości  $\dot{x}_p, \dot{y}_p$ , które wypro-  
wadzane są za pomocą wzorów (3.2), (3.5).

$$\begin{aligned}\dot{x}_p &= \frac{(\omega_l + \omega_r) \cdot D}{2} \cos(\alpha_p) \\ \dot{y}_p &= \frac{(\omega_l + \omega_r) \cdot D}{2} \sin(\alpha_p)\end{aligned}\tag{3.31}$$

Jednakże znajomość równań ruchu oraz pomiarów nie jest wystarczającą metodą wyznaczania dokładnej pozycji. Dzieje się tak ze względu na uproszczenia występujące w modelu oraz błędy odczytów z czujników. Z tego względu w następnym rozdziale opisano rozszerzenie metody pozycjonowania poprzez implementację rozszerzonego filtru Kalmana.

## Rozdział 4

# Rozszerzony filtr Kalmana

### 4.1 Charakterystyka filtru Kalmana

Filtr Kalmana (źródło: [3]) jest rekurencyjnym algorytmem wyznaczania optymalnej estymaty stanu układu dynamicznego, dyskretnego i liniowego. Do tego używana jest predykcja wektora stanu na podstawie modelu matematycznego procesu, a także korekcja otrzymanych wartości wektora z pomocą pomiarów. Zakłada się ponadto, że model i pomiary są obarczone błędem o rozkładzie normalnym. Filtr wykorzystywany jest między innymi w autopilotach, śledzeniu obiektów na podstawie pomiarów z radaru, czy usuwania szumu z nagrań mowy. Ponadto używany jest także w jednej z implementacji algorytmu jednoczesnej lokalizacji i mapowania (EKF-SLAM). Wykorzystywany jest on w robotyce moblinej, w tym także można z niego skorzystać w celu dokładnego określania pozycji robota typu Micromouse w labiryncie i utworzenia mapy środowiska w celu najszybszego dotarcia do środka planszy.

Filtr bazuje na równaniach stanu układu, stąd sam model matematyczny przedstawiany jest w postaci (źródło: [5]):

$$\mathbf{x}_k = \mathbf{A} \cdot \mathbf{x}_{k-1} + \mathbf{B} \cdot \mathbf{u}_{k-1} + \mathbf{w}_{k-1} \quad (4.1)$$

Gdzie:

- $\mathbf{x}_k$ - wektor stanu układu w dyskretniej chwili  $k$
- $\mathbf{x}_{k-1}$ - wektor stanu układu w poprzedniej, dyskretniej chwili  $k-1$
- $\mathbf{w}_{k-1}$ - wektor szumu przetwarzania
- $\mathbf{u}_{k-1}$ - wektor wejściowych/sterujących
- $\mathbf{A}$ - stała macierz stanu układu
- $\mathbf{B}$ - stała macierz wejść układu

Sam model matematyczny procesu jest zwykle obarczony niemierzalnym błędem ( $\mathbf{w}_{k-1}$ ) związanym z przyjmowanymi założeniami upraszczającymi (np. brak poślizgu kół przy poruszaniu się robota), co powoduje, że z czasem niedokładność wyznaczanego stanu wzrasta. Z tego względu filtr łączy tak uzyskany wynik z obliczeniami przeprowadzanymi na podstawie bieżących pomiarów:

$$\mathbf{z}_k = \mathbf{H} \cdot \mathbf{x}_{k-1} + \mathbf{v}_k \quad (4.2)$$

,gdzie:

- $\mathbf{z}_k$ - wektor stanu w bieżącej chwili otrzymany na podstawie pomiarów
- $\mathbf{x}_k$ - chwilowy wektor stanu układu
- $\mathbf{v}_k$ - wektor szumu pomiarowego (niemierzalny)
- $\mathbf{H}$ - stała macierz przekształceń pomiarów

Oczywiście same pomiary także obarczone są niedokładnościami. Z tego względu wektory  $\mathbf{v}_k$  oraz  $\mathbf{w}_{k-1}$  są modelowane za pomocą macierzy kowariancji odpowiednio  $\mathbf{R}$  i  $\mathbf{Q}$ . Wtedy wyniki  $\mathbf{x}_k$  (tutaj oznacza wektor stanu przed uwzględnieniem pomiarów- a priori) oraz  $\mathbf{z}_k$  oznaczają wartości oczekiwane odpowiednio wektora stanu uzyskanego z modelu i na podstawie pomiarów.

Filtr Kalmana bierze pod uwagę oba warianty (modelowe i pomiarowe) i decyduje, który wariant bardziej zdecyduje o wyliczonej wartości wyjściowego wektora estymacji stanu układu (np. jeżeli odchylenie standardowe w wektorze pomiarów jest mniejsze niż w uzyskanym na podstawie modelu, to wynik końcowy bardziej zależeć będzie od tego pierwszego).

## 4.2 Filtr Kalmana dla układów nieliniowych

Standardowo filtr Kalmana przeznaczony jest dla układów liniowych, które dadzą przedstawiać się w postaci równań stanu (4.1). Nie może więc zostać użyty do estymacji położenia robota, którego ruch opisany jest równaniami przedstawionymi w rozdziale 3. Dla systemów nieliniowych implementuje się rozszerzony filtr Kalmana [18, 5], w którym stan systemu w danej chwili przedstawiony jest za pomocą funkcji  $\mathbf{f}$ , zależnej od wektora stanu  $\mathbf{x}_{k-1}$  oraz sterującego  $\mathbf{u}_{k-1}$  z poprzedniej chwili. W podobny sposób wyrażana jest funkcja wyników pomiarów  $\mathbf{h}$  (źródło: [5]):

$$\mathbf{x}_k = \mathbf{f}(\mathbf{x}_{k-1}, \mathbf{u}_{k-1}) + \mathbf{w}_{k-1} \quad (4.3)$$

,gdzie:

- $\mathbf{x}_k$ - wektor stanu w chwili k
- $\mathbf{x}_{k-1}$ - wektor stanu w chwili k-1
- $\mathbf{u}_{k-1}$ - wektor zmiennych sterujących w chwili k-1
- $\mathbf{w}_{k-1}$ - wektor szumu procesowego

$$\mathbf{z}_k = \mathbf{h}(\mathbf{x}_k) + \mathbf{v}_k \quad (4.4)$$

Gdzie:

- $\mathbf{x}_k$ - wektor stanu w chwili k na podstawie pomiarów
- $\mathbf{v}_k$ - wektor szumu pomiarowego

Zarówno funkcja  $\mathbf{f}$ , jak i  $\mathbf{h}$  mogą być nieliniowe, ale wymaganiem jest, aby były klasy  $C^1$ .



### 4.3 Algorytm rozszerzonego filtru Kalmana

Jedna iteracja algorytmu filtru podzielona jest na dwa etapy. W pierwszym (krok predykcji) następuje przewidywanie (predykcja) wartości oczekiwanej wektora stanu a priori  $\mathbf{x}_k^-$ . Ponadto wyliczana jest macierz kowariancji  $\mathbf{P}_k^-$  (także a priori) definiująca błąd estymaty wektora. W drugim etapie (krok korekcji) następuje uwzględnienie estymaty wyliczonej na podstawie pomiarów (wyniki a posteriori) (źródło: [5]).

**Krok predykcji** Określa się w nim przewidywany stan systemu w chwili  $k$ . Na podstawie modelu matematycznego wyliczana jest estymata oraz macierz kowariancji stanu:

$$\mathbf{x}_k^- = \mathbf{f}(\mathbf{x}_{k-1}, \mathbf{u}_{k-1}) \quad (4.5)$$

$$\mathbf{P}_k^- = \mathbf{F}_{k-1} \mathbf{P}_{k-1} \mathbf{F}_{k-1}^T + \mathbf{Q}_{k-1} \quad (4.6)$$

Gdzie:

- $\mathbf{x}_k^-$ - estymowany stan a priori w chwili  $k$
- $\mathbf{x}_{k-1}$ - estymowany stan a posteriori w chwili  $k-1$
- $\mathbf{u}_{k-1}$ - wektor sterujący w chwili  $k-1$
- $\mathbf{P}_k^-$ - estymowana macierz kowariancji stanu a priori w chwili  $k$
- $\mathbf{F}_{k-1}$ - macierz Jacobiego wektora funkcji  $\mathbf{f}$  względem wektora stanu
- $\mathbf{P}_{k-1}$ - estymowana macierz kowariancji stanu a posteriori w chwili  $k-1$
- $\mathbf{Q}_{k-1}$ - macierz kowariancji szumów procesowych

Macierz Jacobiego  $\mathbf{F}$  definiowana jest w następujący sposób:

$$\mathbf{F}_k = \frac{\partial \mathbf{f}}{\partial \mathbf{x}}|_{\mathbf{x}_k, \mathbf{u}_k} = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \dots & \frac{\partial f_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_n}{\partial x_1} & \dots & \frac{\partial f_n}{\partial x_n} \end{bmatrix} \quad (4.7)$$

Natomiast macierz kowariancji szumów  $\mathbf{Q}$ :

$$\mathbf{Q}_k = \begin{bmatrix} E[(x_1 - \mu_1) \cdot (x_1 - \mu_1)] & \dots & E[(x_1 - \mu_1) \cdot (x_n - \mu_n)] \\ \vdots & \ddots & \vdots \\ E[(x_1 - \mu_1) \cdot (x_1 - \mu_1)] & \dots & E[(x_n - \mu_n) \cdot (x_n - \mu_n)] \end{bmatrix} \quad (4.8)$$

Gdzie  $\mu_i$  oznacza wartości oczekiwane zmiennych stanu.

**Krok korekcji** W tym etapie na podstawie występującej odchyłki wartości modelu od odczytanych wartości z czujników  $\mathbf{z}_k - \mathbf{h}(\mathbf{x}_k^-)$  następuje uaktualnienie wartości wektora stanu  $\mathbf{x}_k^-$  oraz macierzy kowariancji  $\mathbf{P}_k^-$ . Otrzymane zostają wartości uwzględniające dokonane pomiary (a posteriori). Kluczowym procesem jest obliczenie wzmocnienia Kalmana  $\mathbf{K}_k$ , która definiuje w każdym kroku w jak znacznym stopniu wartości a priori mają zostać skorygowane:

$$\mathbf{K}_k = \mathbf{P}_k^- \mathbf{H}_k^T (\mathbf{H}_k \mathbf{P}_k^- \mathbf{H}_k^T + \mathbf{R}_k)^{-1} \mathbf{1} \quad (4.9)$$

Gdzie:

- $\mathbf{K}_k$ - wzmocnienie Kalmana w chwili k
- $\mathbf{H}_k$ - macierz Jacobiego wektora funkcji  $\mathbf{h}$  względem wektora stanu
- $\mathbf{R}_k$ - macierz szumu pomiarowego

Macierze  $\mathbf{H}$  oraz  $\mathbf{R}$  są definiowane następująco:

$$\mathbf{H}_k = \frac{\partial \mathbf{h}}{\partial \mathbf{x}}|_{\mathbf{x}_k^-} = \begin{bmatrix} \frac{\partial h_1}{\partial x_1} & \cdots & \frac{\partial h_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial h_n}{\partial x_1} & \cdots & \frac{\partial h_n}{\partial x_n} \end{bmatrix} \quad (4.10)$$

$$\mathbf{R}_k = \begin{bmatrix} E[(z_1 - \mu_1) \cdot (z_1 - \mu_1)] & \cdots & E[(z_1 - \mu_1) \cdot (z_n - \mu_n)] \\ \vdots & \ddots & \vdots \\ E[(z_1 - \mu_1) \cdot (z_1 - \mu_1)] & \cdots & E[(z_n - \mu_n) \cdot (z_n - \mu_n)] \end{bmatrix} \quad (4.11)$$

Uzyskawszy macierz wzmocnienia Kalmana można obliczyć wartości a posteriori wektora stanu  $\mathbf{x}_k$  oraz macierzy kowariancji  $\mathbf{P}_k$ :

$$\mathbf{x}_k = \mathbf{x}_k^- + \mathbf{K}_k(\mathbf{z}_k - \mathbf{h}(\mathbf{x}_k^-)) \quad (4.12)$$

$$\mathbf{P}_k = (\mathbf{I} - \mathbf{K}_k \mathbf{H}_k) \mathbf{P}_k^- \quad (4.13)$$

## 4.4 Implementacja filtru dla robota Micromouse

Łącząc model kinematyczny robota o napędzie różnicowym wyprowadzony w rozdziale 3 oraz pomiary uzyskiwane z enkoderów, żyroskopu oraz akcelerometru można użyć rozszerzonego filtru Kalmana do lokalizacji w labiryncie. W tym celu obrany został wektor stanu zawierający zmienne estymowane przez algorytm:

$$\mathbf{x}(\mathbf{t}) = \begin{bmatrix} x \\ y \\ \alpha \\ \omega \end{bmatrix} \quad (4.14)$$

,gdzie:

- $x$ - przemieszczenie liniowe robota względem osi x układu globalnego  $\pi_0$

- $y$ - przemieszczenie liniowe robota względem osi  $y$  układu globalnego  $\pi_0$
- $\alpha$ - kąt obrotu układu  $\pi_1$  związanego z robotem względem układu globalnego  $\pi_0$
- $\omega$ - wartość prędkości kątowej obrotu robota

Zmienne  $x, y, \alpha$  są niezbędne w celu jednoznacznego wyznaczenia pozycji konstrukcji w układzie labiryntu. Dodatkowa estymacja wartości prędkości kątowej obrotu robota pozwala na jeszcze większe zmniejszenie wpływu dryfu żyroskopu na pomiary.

**Predykcja** Zależności modelu kinematyki na wektor położenia (3.23) oraz (3.29) w przypadku gdy  $\omega = 0$  bezpośrednio wyznaczają funkcje kroku predykcji:

$$\mathbf{f}(\mathbf{t}) = \begin{bmatrix} x(t) \\ y(t) \\ \alpha(t) \\ \omega(t) \end{bmatrix} = \begin{bmatrix} \frac{1}{\omega_p^2} \cdot (-a_x \cos(\alpha_p + \omega_p \cdot \Delta t) + a_x \cos(\alpha_p) + a_y \sin(\alpha_p + \omega_p \cdot \Delta t) - a_y \cos(\alpha_p)) \\ \frac{1}{\omega_p^2} \cdot (-a_x \sin(\alpha_p + \omega_p \cdot \Delta t) + a_x \sin(\alpha_p) - a_y \cos(\alpha_p + \omega_p \cdot \Delta t) + a_y \cos(\alpha_p)) \\ \omega_p \cdot \Delta t \\ 0 \end{bmatrix} + \begin{bmatrix} (\dot{x}_p + \frac{1}{\omega_p} \cdot (-a_x \sin(\alpha_p) + a_y \cos(\alpha_p)) \cdot \Delta t \\ (\dot{y}_p + \frac{1}{\omega_p} \cdot (a_x \cos(\alpha_p) - a_y \sin(\alpha_p)) \cdot \Delta t \\ 0 \\ 0 \end{bmatrix} + \begin{bmatrix} x_p \\ y_p \\ \alpha_p \\ \omega_p \end{bmatrix} \quad (4.15)$$

Dla  $\omega = 0$ :

$$\mathbf{f}(t) = \begin{bmatrix} x(t) \\ y(t) \\ \alpha(t) \\ \omega(t) \end{bmatrix} = \begin{bmatrix} (a_x \cos(\alpha_p) - a_y \sin(\alpha_p)) \cdot \frac{\Delta t^2}{2} + \dot{x}_p \cdot \Delta t + x_p \\ (a_x \sin(\alpha_p) + a_y \cos(\alpha_p)) \cdot \frac{\Delta t^2}{2} + \dot{y}_p \cdot t + y_p \\ \alpha_p \\ \omega_p \end{bmatrix} \quad (4.16)$$

Znając funkcję  $\mathbf{f}(\mathbf{t})$  można wyznaczyć jej macierz Jacobiego. W celu przejrzystości zapisu po-

chodne cząstkowe wyprowadzane będą wierszami (indeks dolny funkcji oznacza różniczkowane równanie):

$$\begin{aligned}
\frac{\partial f_1}{\partial x} &= 1 \\
\frac{\partial f_1}{\partial y} &= 0 \\
\frac{\partial f_1}{\partial \alpha} &= \frac{a_x}{\omega_p^2} \cdot \sin(ap + \omega_p \cdot \Delta t) + \frac{a_y}{\omega_p^2} \cdot \cos(ap + \omega_p \cdot \Delta t) \\
\frac{\partial f_1}{\partial \omega} &= 0
\end{aligned} \tag{4.17}$$

$$\begin{aligned}
\frac{\partial f_2}{\partial x} &= 0 \\
\frac{\partial f_2}{\partial y} &= 1 \\
\frac{\partial f_2}{\partial \alpha} &= \frac{a_x}{\omega_p^2} \cdot \sin(ap + \omega_p \cdot \Delta t) + \frac{a_y}{\omega_p^2} \cdot \cos(ap + \omega_p \cdot \Delta t) \\
\frac{\partial f_2}{\partial \omega} &= 0
\end{aligned} \tag{4.18}$$

$$\begin{aligned}
\frac{\partial f_3}{\partial x} &= 0 \\
\frac{\partial f_3}{\partial y} &= 0 \\
\frac{\partial f_3}{\partial \alpha} &= 1 \\
\frac{\partial f_3}{\partial \omega} &= 0
\end{aligned} \tag{4.19}$$

$$\begin{aligned}
\frac{\partial f_4}{\partial x} &= 0 \\
\frac{\partial f_4}{\partial y} &= 0 \\
\frac{\partial f_4}{\partial \alpha} &= 0 \\
\frac{\partial f_4}{\partial \omega} &= 1
\end{aligned} \tag{4.20}$$

W przypadku, gdy  $\omega = 0$ :

$$\begin{aligned}
\frac{\partial f_1}{\partial x} &= 1 \\
\frac{\partial f_1}{\partial y} &= 0 \\
\frac{\partial f_1}{\partial \alpha} &= 0 \\
\frac{\partial f_1}{\partial \omega} &= 0
\end{aligned} \tag{4.21}$$

$$\begin{aligned}
\frac{\partial f_2}{\partial x} &= 0 \\
\frac{\partial f_2}{\partial y} &= 1 \\
\frac{\partial f_2}{\partial \alpha} &= 0 \\
\frac{\partial f_2}{\partial \omega} &= 0
\end{aligned} \tag{4.22}$$

$$\begin{aligned}
\frac{\partial f_3}{\partial x} &= 0 \\
\frac{\partial f_3}{\partial y} &= 0 \\
\frac{\partial f_3}{\partial \alpha} &= 1 \\
\frac{\partial f_3}{\partial \omega} &= 0
\end{aligned} \tag{4.23}$$

$$\begin{aligned}
\frac{\partial f_4}{\partial x} &= 0 \\
\frac{\partial f_4}{\partial y} &= 0 \\
\frac{\partial f_4}{\partial \alpha} &= 0 \\
\frac{\partial f_4}{\partial \omega} &= 1
\end{aligned} \tag{4.24}$$

Pochodne cząstkowe (4.17)-(4.24) definiują macierz Jacobiego  $\mathbf{F}$ .

**Pomiary** Wektor pomiarów  $\mathbf{z}(\mathbf{t})$  uzyskiwany jest poprzez odpowiednie przekształcenia zebranych z czujników danych i jest zdefiniowany jako:

$$\mathbf{z}(\mathbf{t}) = \begin{bmatrix} x_{pomiar}(t) \\ y_{pomiar}(t) \\ \alpha_{pomiar}(t) \\ \omega_{pomiar}(t) \end{bmatrix} \tag{4.25}$$

Gdzie:

- $x_{pomiar}(t)$  określa przemieszczenie wzdłuż osi x układu  $\pi_0$  na podstawie pomiarów z enkoderów, korzystając przy tym z równań (3.5) oraz (3.9)
- $y_{pomiar}(t)$  określa przemieszczenie wzdłuż osi y układu  $\pi_0$  na podstawie pomiarów z enkoderów, korzystając przy tym z równań (3.5) oraz (3.9):
- $\alpha_{pomiar}(t)$  określa orientację w układzie  $\pi_0$  na podstawie pomiarów z enkoderów, korzystając ze wzoru (3.6).
- $\omega_{pomiar}(t)$  pomiar uzyskiwany bezpośrednio z żyroskopu

Ze względu na obrany wektor pomiarów (na podstawie równania (4.4)) można zauważyć, że  $\mathbf{h}(\mathbf{x}_k) = x_k$ . Stąd macierz Jacobiego  $\mathbf{H}$  jest zdefiniowana jak poniżej:

$$\mathbf{H}_k = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (4.26)$$

Zastosowanie filtru Kalmana wymaga jeszcze wyznaczenia macierzy kowariancji  $\mathbf{Q}$  szumu procesowego oraz  $\mathbf{R}$  szumu pomiarowego.

**Macierz  $\mathbf{Q}$**  jest ze względu na obrany model macierzą diagonalną. Wartości wyznaczone zostały na podstawie doświadczeń w działaniu filtru poprzez badanie błędów wyznaczania pozycji podczas przejazdu dla każdych dobranych wartości odchyleń standardowych:

$$\mathbf{Q}_k = \begin{bmatrix} 0.003[mm]^2 & 0 & 0 & 0 \\ 0 & 0.003[mm]^2 & 0 & 0 \\ 0 & 0 & 0.003[^\circ]^2 & 0 \\ 0 & 0 & 0 & 0.003[\frac{^\circ}{s}]^2 \end{bmatrix} \quad (4.27)$$

## 4.5 Testy rozwiązania

Testy miały na celu sprawdzenie dokładności wyznaczania pozycji robota. Aby sprawdzić zastosowane rozwiązanie wykonywane były zaplanowane przejazdy. Wyniki uzyskiwane z obliczeń filtru Kalmana porównywane były z wynikami z pomiarów rzeczywistego przemieszczenia. Testy przeprowadzono do zbadania poprawności wyznaczania orientacji robota oraz przemieszczenia wzdłuż linii prostej.

### 4.5.1 Testy orientacji

Test był wykonywany w celu sprawdzenia dokładności, z jaką filtr wyznacza orientację konstrukcji na podstawie pomiarów z żyroskopu.

Eksperyment polegał na wykonaniu 4 następujących po sobie obrotów robota o  $90^\circ$ , zaczynając od orientacji wyjściowej  $0^\circ$ . Po każdym obrocie sprawdzany był pomiar orientacji wskazywany przez robota oraz mierzony był dokładny kąt o jaki obrócono konstrukcję. Wyniki przedstawiono w tabeli 4.1:

Tablica 4.1: Pomiary orientacji robota

Nr pomiaru	Wartość dokładna $[\circ]$	Wartość wskazywana przez robota $[\circ]$	Błąd orientacji $[\circ]$
1	0	1.15	1.15
2	89	89.86	0.86
3	182	180.78	1.22
4	271	269.12	1.88
5	359	357.81	1.19

Na podstawie danych uzyskanych z powyższego eksperymentu widoczny jest brak wpływu dryfu żyroskopu na wyniki pomiaru (błąd nie ulega narastaniu). Ponadto estymowana wartość

kąta nie odbiega znacząco od wartości dokładnej. Średnia wartość błędu wyniosła  $1.26^\circ$ . Przy długim przejeździe robota niedokładność ta może powodować błędy pomiaru pozycji.

#### 4.5.2 Testy jazdy po prostej

Test był wykonywany w celu sprawdzenia dokładności, z jaką filtr wyznacza położenie robota.

Eksperyment polegał na wykonaniu przejazdu konstrukcji po linii prostej i zatrzymywaniu się co  $25\text{cm}$ . Przy każdym zatrzymaniu sprawdzana była wskazywana przebyta odległość, która porównywana była z rzeczywistą, zmierzoną odległością od miejsca startu. Wyniki przedstawia poniższa tabela 4.2:

Tablica 4.2: Pomiary przejazdu po prostej

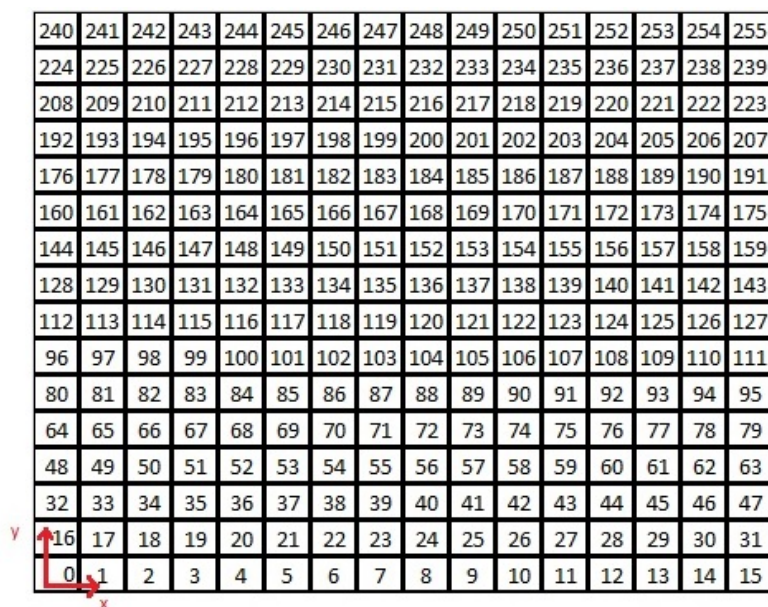
Nr pomiaru	Wartość dokładna [mm]	Wartość wskazywana przez robota [mm]	Błąd pozycji [mm]
1	0	9	9
2	252	237	15
3	503	485	18
4	752	731	21
5	998	973	25

Kolejne wyniki wskazują na to, że błąd wyznaczania pozycji rośnie wraz z przebytą przez robota drogą. Niedokładność pomiaru odległości nie jest duża, jednakże podczas długiego przejazdu może mieć znaczący wpływ na odczyt aktualnej pozycji. Stąd dobrym rozwiązaniem byłoby korygowanie aktualnego pomiaru pozycji poprzez sprawdzanie odległości robota od najbliższej ścianki na wprost za pomocą czujników odległości.

## Rozdział 5

# Algorytm wyszukiwania drogi w labiryncie

Mając możliwość wyznaczenia dokładnego położenia w układzie związanym z labiryntem możliwe staje się wskazanie, w której komórce znajduje się w tym momencie robot. Numeracja komórek oraz położenie układu zostało przyjęte jak na rysunku 5.1:



240	241	242	243	244	245	246	247	248	249	250	251	252	253	254	255
224	225	226	227	228	229	230	231	232	233	234	235	236	237	238	239
208	209	210	211	212	213	214	215	216	217	218	219	220	221	222	223
192	193	194	195	196	197	198	199	200	201	202	203	204	205	206	207
176	177	178	179	180	181	182	183	184	185	186	187	188	189	190	191
160	161	162	163	164	165	166	167	168	169	170	171	172	173	174	175
144	145	146	147	148	149	150	151	152	153	154	155	156	157	158	159
128	129	130	131	132	133	134	135	136	137	138	139	140	141	142	143
112	113	114	115	116	117	118	119	120	121	122	123	124	125	126	127
96	97	98	99	100	101	102	103	104	105	106	107	108	109	110	111
80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95
64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79
48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

Rysunek 5.1: Numeracja pól labiryntu (źródło: materiały autora)

Układ  $\pi_0$  związany jest ze środkiem komórki o numerze 0. Ze względu na typową dla języka C indeksację tablic od 0 numeracja pól labiryntu także zaczyna się od 0. Zakładane jest także, iż robot zaczyna przeszukiwanie zwrócony w kierunku osi  $y$  oraz, że znajduje się na początku w komórce 0. Celem konstrukcji jest dojechanie do któregośkolwiek z pól z numerem 119, 120, 135, 136 (środek labiryntu).

Pozycja robota wyznaczana jest jako przemieszczenie (w mm) wzdłuż osi  $x$  oraz  $y$  względem początku układu współrzędnych. Wiedząc także że odległość pomiędzy sąsiadującymi komórkami wynosi 18 mm można wyznaczyć, w którym polu (o jakim numerze) znajduje się robot:

$$NUMERKOMORKI = \left\lfloor \frac{x}{18} \right\rfloor + \left\lfloor \frac{y}{18} \right\rfloor \cdot 16 \quad (5.1)$$



Przejazd o jedną komórkę wzdłuż osi  $Y$  układu związanego z labiryntem powoduje wzrost indeksu wskazywanego pola o 16. Stąd wynik operacji  $\lfloor \frac{y}{18} \rfloor$  jest dodatkowo mnożony przez 16. Zakładane jest ponadto, że jazda robota odbywa się po liniach prostych w czterech kierunkach:

- EAST- zgodny ze zwrotem osi  $x$
- NORTH- zgodny ze zwrotem osi  $y$
- WEST- ze zwrotem przeciwnym niż osi  $x$
- SOUTH- ze zwrotem przeciwnym niż osi  $y$

Autor opracował funkcję w języku C wyznaczającą tak zdefiniowaną pozycję i orientację. Poniżej przedstawiony został jej kod:

---

```
void actualPosition(float posx, float posy, float alpha) {
    MAP.labyrinthPosition = (((int) (posy * 100)) / NODE_SEPARATION)
        * LABYRINTH_LENGTH
        + (((int) (posx * 100)) / NODE_SEPARATION) * LABYRINTH_LENGTH;
    int degrees = alpha * 180/M_PI;
    if(degrees < 315 && degrees >= 225)
        MAP.direction = SOUTH;
    else if(degrees >= 315 || degrees < 45)
        MAP.direction = EAST;
    else if(degrees >= 45 && degrees < 135)
        MAP.direction = NORTH;
    else if(degrees >= 135 && degrees < 225)
        MAP.direction = WEST;
}
```

---

Funkcja przyjmuje trzy zmienne opisujące pozycję (w mm) oraz orientację robota (w radianach) i na tej podstawie aktualizuje pola struktury MAP, o której więcej powiedziane będzie w sekcji poświęconej mapie w pamięci robota 5.2.1. Makra `NODE_SEPARATION`, `LABYRINTH_LENGTH`, a także dotyczące kierunków jazdy zdefiniowane są następująco:

---

```
#define NODE_SEPARATION 18 //[mm]

#define LABYRINTH_LENGTH 16 // ilosc komorek w wierszu labiryntu

#define NORTH 2
#define EAST 1
#define SOUTH 0
#define WEST 3
```

---

## 5.1 Sposoby przeszukiwania labiryntu

W sekcji przedstawione zostaną najpopularniejsze algorytmy wykorzystywane w zawodach robotów Micromouse.

### 5.1.1 Metoda podążania wzdłuż ściany

Polega ona na przejeździe labiryntu podążając ciągle wzdłuż wybranej ściany labiryntu (lewej, bądź prawej) (źródło: [8]). Poniżej przedstawiono prosty algorytm zapisany w pseudokodzie, realizujący metodę:

---

```
1. JEZELI(mozna_skrecic_w_prawo)
    Skrec w prawo.
    Wykonaj 5.
2. JEZELI(mozna_jechac_prosto)
    Wykonaj 5.
3. JEZELI(mozna_skrecic_w_lewo)
    Skrec w lewo.
    Wykonaj 5.
4. Zawroc.
5. JEZELI(Pole != CEL)
    Wykonaj 1
    W PRZECIWNYM RAZIE
    Stop.
```

---

W ten prosty sposób możliwe jest dojechanie do środka labiryntu, jednakże znaleziona droga często nie jest drogą najkrótszą. Ponadto algorytm działa jedynie wtedy, gdy pola wyznaczające cel robota są połączone z brzegiem labiryntu. Z tego względu na zawodach Micromouse stosowane są często trasy niespełniające tego wymagania, co uniemożliwia wykorzystanie tej metody. Na rysunku 5.2 pokazano przykładowy labirynt, który nie może zostać rozwiązany metodą przejazdu wzdłuż prawej ściany.

### 5.1.2 Metoda wymuszania brute force

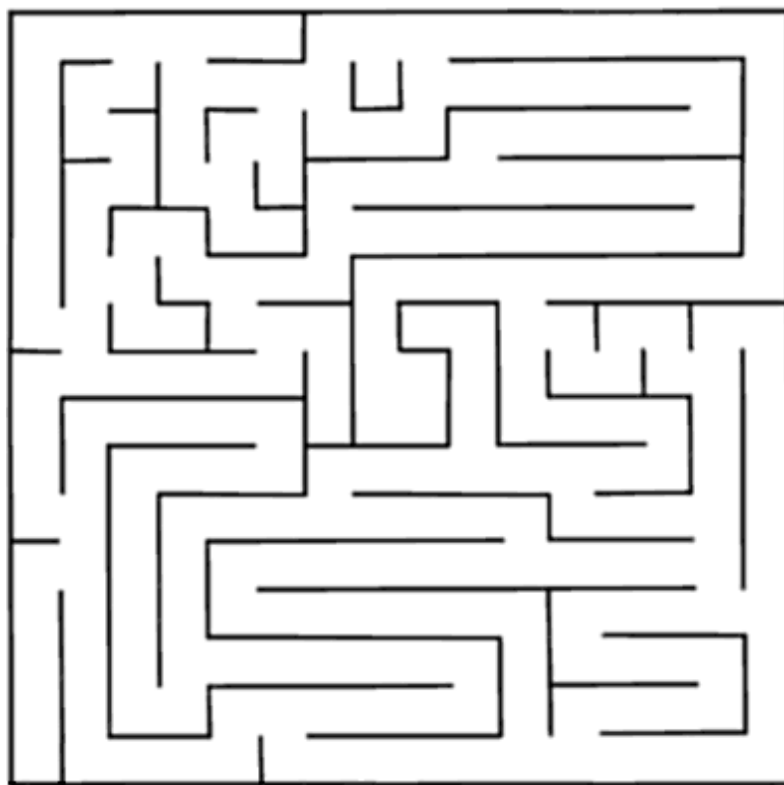
Metoda polega na jeździe po nieodwiedzonych polach tak długo, aż nie zostanie odnalezione jedno z pól centralnych. Wymaga więc zapamiętywania pól raz już odwiedzonych. Cały algorytm postępowania można zapisać jako (źródło: [8]):

---

```
1. Jedz przed siebie.
2. JEZELI(zakret)
    Skrec w losowym, nieodwiedzonym kierunku.
3. JEZELI(slepa_uliczka)
    Wroc na zakret.
4. JEZELI(Pole != CEL)
    Wykonaj 1
    W PRZECIWNYM RAZIE
    Stop.
```

---

Metoda ta w przeciwieństwie do poprzedniej zawsze pozwala na znalezienie drogi do środka labiryntu, jednakże daje małą szansę, że znaleziona ścieżka będzie najkrótszą z możliwych.



Rysunek 5.2: Schemat labiryntu, w którym nie działa metoda prawej strony (źródło: [8])

### 5.1.3 Metody grafowe-algorytm Bellmana Forda

Algorytm Bellmana-Forda (źródło: [1]) służy do wyznaczania najkrótszych ścieżek w grafie z jednego wierzchołka (zwanego wierzchołkiem źródłowym) do pozostałych. Algorytm może być również wykorzystywany do sprawdzania, czy w grafie występują ujemne cykle. W trakcie wykonywania algorytmu dla każdego wierzchołka zostają wyznaczone dwie wartości: koszt dotarcia do tego wierzchołka (oznaczony jako  $d_i$ ) oraz poprzedni wierzchołek na ścieżce ( $p_i$ ). Na początku działania algorytmu dla wierzchołka źródłowego koszt dotarcia wynosi 0, a dla każdego innego wierzchołka nieskończoność. Następnie dla każdej krawędzi (aktualnie analizowana krawędź ma wagę oznaczoną  $k$  i prowadzi z wierzchołka  $U$  do wierzchołka  $V$ ) wykonujemy następującą czynność:

Jeżeli  $d(V) > d(U) + k$ , to ustawiana jest wartość  $d(V)$  na  $d(U) + k$ , a wartość  $p(V)$  na  $U$ .

Całość (przejrzenie wszystkich krawędzi) należy powtórzyć  $n-1$  razy, gdzie  $n$  jest liczbą wierzchołków. W każdej iteracji należy przejrzeć wszystkie krawędzie w tej samej kolejności. Jeśli w którejś iteracji nie nastąpią żadne zmiany, wykonywanie algorytmu można przerwać wcześniej.

Ze względu na charakterystykę labiryntu możliwe jest przedstawienie go jako spójnego grafu ważonego. Wtedy komórki przyjmują postać wierzchołków grafu, natomiast możliwość przejazdu pomiędzy sąsiadującymi polami oznaczana jest krawędziami grafu o odpowiednich wagach.

Przedstawiając labirynt w ten sposób znalezienie najkrótszej drogi sprowadza się do rozwiązania problemu najkrótszej ścieżki w grafie ważonym pomiędzy wierzchołkami o numerach 0 oraz 119, 120, 135 lub 136 (rysunek 5.1). Do tego zadania wykorzystywany jest właśnie algorytm Bellmana-Forda.

Aby przedstawić algorytm w postaci procedur przyjmowane są poniższe oznaczenia (źródła: [11, 16]). Mając dany graf  $G = (V, E)$ , w którym wagi krawędzi przypisywane są na podstawie funkcji  $w : E \rightarrow \mathbb{R}$  wagę ścieżki  $k$  (możliwej drogi) z wierzchołka  $v_0$  do wierzchołka  $v_k$  można przedstawić jako:

$$w(k) = \sum_{i=0}^{k-1} w(v_i, v_{i+1}) \quad (5.2)$$

Ponadto przez  $s$  oznaczony jest wierzchołek źródłowy, z którego liczona jest odległość. Wtedy oznaczając  $d(v)$  jako funkcję wagi ścieżki od wierzchołka  $s$  do  $v$  można użyć stosowanej w algorytmie Bellmana-Forda metody relaksacji krawędzi. Dla każdego wierzchołka  $v \in V$  przypisywany jest także wierzchołek poprzedzający, przez który należy przejść, aby wskazaną ścieżką dojść do wierzchołka  $v$ :  $p(v)$ . Rozpoczęcie działania algorytmu zaczyna się od inicjalizacji wartości początkowych:

---

INICJALIZACJA( $G, s$ )

1. DLA KAZDEGO  $v \in V$  WYKONAJ:

$d(v) = \infty$

$p(v) = \text{NULL}$

2.  $d(s) = 0$

3. ZWROC ( $d, p$ )

---

Relaksacja krawędzi grafu polega na sprawdzeniu, czy przechodząc przez krawędź  $(u, v)$  nie znajdzie się krótszej ścieżki z  $s$  do  $v$  niż dotychczas. Całe postępowanie przedstawia procedura RELAKSACJA:

---

RELAKSACJA( $u, v, w, d, p$ )

1. JEZELI  $d(v) > d(u) + w(u, v)$ :

$d(v) = d(u) + w(u, v)$

$p(v) = u$

---

Z pomocą powyższej procedury można zapisać algorytm Bellmana-Forda jako:

---

BELLMAN-FORD( $G, w, s$ )

1. ( $d, p$ ) = INICJUA( $G, s$ )

2 DLA KAZDEGO  $i=1$  DO  $|V|-1$  WYKONAJ:

DLA KAZDEGO  $(u, v) \in E$  WYKONAJ:

RELAKSACJA( $u, v, w, d, p$ )

3. DLA KAZDEGO  $(u, v) \in E$  WYKONAJ:

JEZELI  $d(v) > d(u) + w(u, v)$ :

ZWROC NULL

4. ZWROC ( $d, p$ )

---

Cechą charakterystyczną algorytmu Bellmana-Forda jest to, że działa on także w grafach, w których wagi krawędzi mogą przyjmować wartości ujemne. Jedynym warunkiem znalezienia rozwiązania jest brak cykli o ujemnej wadze, które w procedurze wykrywane są za pomocą instrukcji nr 3 (zwracana jest wtedy wartość NULL). Złożoność czasowa metody wynosi  $O(|V| \cdot |E|)$ .

## 5.2 Implementacja algorytmu Bellmana-Forda

### 5.2.1 Mapa w pamięci robota

Ze względu na to, że labirynt musi zostać sprowadzony do postaci grafu potrzebny jest sposób na jego implementację w pamięci mikrokontrolera, której wielkość jest bardzo ograniczona. Z tego względu standardowe metody pozwalające na zapisanie struktury grafu (źródło: [20]) w programie mogą być zbyt kosztowne. Z trzech sposobów, jakimi są macierz sąsiedztwa, macierz incydencji oraz lista sąsiedztwa jedynym rozsądnym wydaje się ten ostatni, gdyż pozwala na pominięcie w pamięci nieistniejących krawędzi. Jednakże ze względu na to, że na początku nie znany jest rozkład ścian labiryntu oraz przyjmowany jest stan labiryntu bez ścian (sąsiednie komórki są ze sobą połączone), miejsce potrzebne na przechowanie takiej listy nadal jest duże (tablica 256-elementowa, każdy element stanowi listę 4-elementową). W związku z tym (wykorzystując szczególną specyfikę labiryntu) zastosowano szczególny sposób implementacji grafu w pamięci programu.

Wiadomo, że maksymalna liczba krawędzi wychodząca z wierzchołka w przypadku grafu reprezentującego labirynt wynosi 4. Fakt ten daje możliwość zachowania informacji o występujących w danej komórce ściankach w 1 bajcie. W programie zastosowano przedstawienie grafu w postaci tablicy 256-elementowej, w której każdy element to 1-bajtowa zmienna typu całkowitoliczbowego bez znaku (`uint8_t`):

$$--PNESW \quad (5.3)$$

Gdzie:

- bit  $P$  mówi, czy komórka była już odwiedzona (1), czy nie (0)
- bit  $N$  mówi, czy z komórki da się przejechać w kierunku północnym (w górę)- bit ma wartość 1, gdy jest ścianka
- bit  $E$  mówi, czy z komórki da się przejechać w kierunku wschodnim (w prawo)- bit ma wartość 1, gdy jest ścianka
- bit  $S$  mówi, czy z komórki da się przejechać w kierunku południowym (w dół)- bit ma wartość 1, gdy jest ścianka
- bit  $W$  mówi, czy z komórki da się przejechać w kierunku zachodnim (w lewo)- bit ma wartość 1, gdy jest ścianka
- 3 najstarsze bity nie mają zastosowania

Struktura MAP 5.2.1 służy do przechowywania informacji o ułożeniu labiryntu oraz aktualnym położeniu robota. Tablica opisująca konfigurację ścianek nosi nazwę *labyrinth*:

---

```
struct LOCALIZATION{
    uint8_t labyrinth[256]; // tablica scianek
    int value[256]; // waga komorki
    uint8_t labyrinthPosition; // numer komorki
    uint8_t direction; //kierunek robota
    uint8_t setDirection; // docelowy kierunek robota
```

```

uint8_t destinationCell; //docelowa komorka w danej chwili
uint8_t weight; //waga przejazdu z jednej komorki do drugiej
uint8_t step; //etap przejazdu 0- mapowanie, 1- powrot do poczatku, 2- przejazd do
srodka
}MAP;

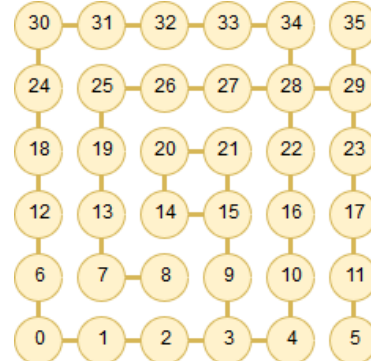
```

---

Poniżej przedstawiono przykładowy labirynt 6x6 oraz jego reprezentację grafową:

30	31	32	33	34	35
24	25	26	27	28	29
18	19	20	21	22	23
12	13	14	15	16	17
6	7	8	9	10	11
0	1	2	3	4	5

(a) Przykładowy labirynt 6x6



(b) Reprezentacja labiryntu w postaci grafu

Rysunek 5.3: Przykład reprezentacji labiryntu za pomocą grafu (źródło: materiały autora)

### 5.2.2 Działanie algorytmu

Na początku przejazdu robot nie ma pojęcia, w jaki sposób ułożone są ściany (oprócz komórki oznaczonej jako 0 - tam się znajduje, gdy startuje). Przyjmowane jest założenie, że konstrukcja skierowana jest w kierunku północnym oraz wewnątrz labiryntu nie ma ścianek. Ponadto wagi 4 komórek docelowych ustawiane są jako 0 (są one zapamiętywane w tablicy *value*), natomiast wagi pozostałych przyjmują wartość zdefiniowaną jako *MAX\_PATH*:

```
#define MAX_PATH 512
```

---

Ważne jest, aby wartość ta była duża, większa niż maksymalna możliwa w labiryncie długość ścieżki do wierzchołków docelowych. Jest ona odpowiednikiem  $\infty$  w algorytmie Bellmana. Podczas etapu mapowania wagi wszystkich krawędzi ustalane są jako 1.

Poniżej przedstawiono funkcję *BellmanFord()* będącą implementacją algorytmu dla problemu robota Micromouse. Jest ona wykorzystywana w programie ilekroć następuje aktualizacja mapy labiryntu, bądź konstrukcja ma za zadanie dojechać do wyznaczonej komórki:

```

//Funkcja implementacji algorytmu Bellmana-Forda do problemu
//rozwiązywanego przez robota. Jako argumenty przyjmowany jest
//numer komórki docelowej (destination):
void BellmanFord(uint8_t destination) {
    //Rozmiar labiryntu:
    int N = LABYRINTH_LENGTH * LABYRINTH_LENGTH;
    //Inicjalizacja wartosci:
    for (int i = 0; i < N; ++i)
        MAP.value[i] = MAX_PATH;
    //Jezeli poczatkiem jest ktoras ze srodkowych:
    int middle = (LABYRINTH_LENGTH + 1) * (LABYRINTH_LENGTH / 2 - 1);

```

```

if (destination == middle || destination == middle + 1
    || destination == middle + LABYRINTH_LENGTH
    || destination == middle + LABYRINTH_LENGTH + 1) {
    MAP.value[middle] = 0;
    MAP.value[middle + 1] = 0;
    MAP.value[middle + LABYRINTH_LENGTH] = 0;
    MAP.value[middle + LABYRINTH_LENGTH + 1] = 0;
} else
    MAP.value[destination] = 0;
//Po inicjalizacji następuje wykonanie algorytmu.
for (int k = 1; k < N; ++k) {
    //Przeszukiwanie kazdego wierzchołka:
    for (int i = 0; i < N; ++i) {
        //Przeszukiwanie 4 mozliwych krawedzi wierzchołka:
        //Kierunek NORTH:
        if (i + LABYRINTH_LENGTH < N) {
            if (!(MAP.labyrinth[i] & 0x08)
                && MAP.value[i] < MAP.value[i + LABYRINTH_LENGTH] + MAP.weight)
                MAP.value[i] = MAP.value[i + LABYRINTH_LENGTH] + MAP.weight;
        }
        //Kierunek EAST:
        if ((i + 1) % LABYRINTH_LENGTH != 0) {
            if (!(MAP.labyrinth[i] & 0x04)
                && MAP.value[i] < MAP.value[i + 1] + MAP.weight)
                MAP.value[i] = MAP.value[i + 1] + MAP.weight;
        }
        //Kierunek SOUTH:
        if (i - LABYRINTH_LENGTH > 0) {
            if (!(MAP.labyrinth[i] & 0x02)
                && MAP.value[i] < MAP.value[i - LABYRINTH_LENGTH] + MAP.weight)
                MAP.value[i] = MAP.value[i - LABYRINTH_LENGTH] + MAP.weight;
        }
        //Kierunek WEST:
        if ((i - 1) % LABYRINTH_LENGTH != LABYRINTH_LENGTH - 1) {
            if (!(MAP.labyrinth[i] & 0x01)
                && MAP.value[i] < MAP.value[i - 1] + MAP.weight)
                MAP.value[i] = MAP.value[i - 1] + MAP.weight;
        }
    }
}
}
}

```

---

**Mapowanie** Ze względu na nieznaną formę labiryntu na początku wykonywany jest przejazd mający na celu utworzenie w pamięci robota mapy, dzięki której możliwe będzie wyznaczenie najkrótszej ścieżki. Po wejściu do nieodwiedzanej komórki następuje aktualizacja ścianek labiryntu.

Funkcja *mapping()* służy do oznaczania istniejących ścianek na podstawie odczytów z czujników odległości gdy robot znajduje się w środku komórki. Wartości odczytów *measurements[]*

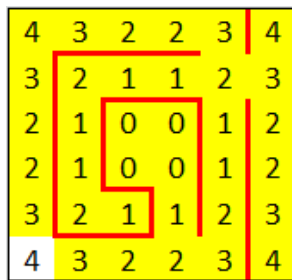
porównywane są z progiem *sensorThreshold* zdefiniowaną w programie. Jeżeli są one mniejsze niż wartość referencyjna oznacza to, że dany czujnik wykrył ścianę:

---

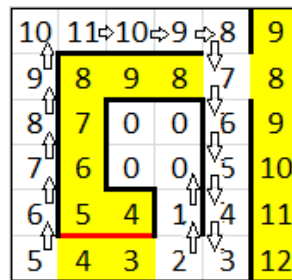
```
//Funkcja mapujaca- przyjmuje jako argument wskaznik do
//tablicy 3-elementowej measurement[3] pomiarow z czujnikow
//odleglosci odpowiednio po lewej, na wprost i po prawej
//stronie robota:
void mapping(int* measurement) {
    //Sprawdzanie czy dojechano do docelowej komorki
    //i czy nie byla ona odwiedzona
    //(iloczyn bitowy bitu mowiacego, czy komorka byla odwiedzona ---P---- z 00010000):
    if (MAP.labyrinthPosition == MAP.destinationCell
        && !(MAP.labyrinth[MAP.labyrinthPosition & 0x1F])) {
        //Jezeli komorka nie byla jeszcze mapowana
        //wpisywane sa odpowiednie scianki:
        for (int i = 0; i < 3; ++i) {
            if (measurement[i] < sensorThreshold) {
                MAP.labyrinth[MAP.labyrinthPosition] =
                    (1 << ((MAP.direction + i) % 4)) |
                    MAP.labyrinth[MAP.labyrinthPosition];
                //Uwzgledniany jest aktualnie obrany kierunek
                //robota (MAP.direction).
            }
        }
    }
}
```

---

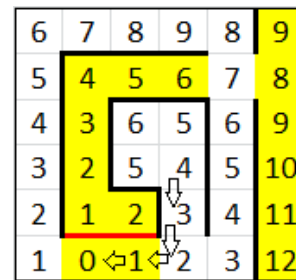
Rysunek 5.5 przedstawia proces mapowania dla przykładowego labiryntu wspomnianego wcześniej (5.3a):



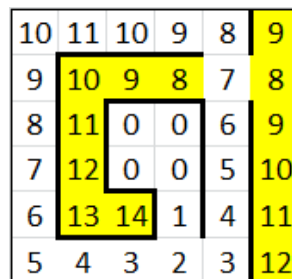
(a) Początek mapowania labiryntu



(b) Mapowanie aż do dotarcia do środka



(c) Mapowanie pozostałych komórek



(d) Koniec mapowania

Rysunek 5.4: Przykładowy przejazd mapujący labirynt 6x6 (czerwone krawędzie oznaczają nieodkryte ściany, żółte pola nieodwiedzone komórki) (źródło: materiały autora)



Jeżeli nie zaobserwowano żadnej zmiany, robot jedzie do następnego nieodwiedzonego pola. W przeciwnym wypadku zostaje wyznaczona za pomocą algorytmu Bellmana-Forda najkrótsza ścieżka do środka przy danym, zaktualizowanym stanie. Następna komórka wybierana jest po najmniejszej wadze. Z tego względu gdy robot natknie się na ślepy zaułek, po aktualizacji wag obierze kierunek powrotny, gdyż w tamtą stronę będą one maleć. Taki schemat przeszukiwania trwa do momentu znalezienia dowolnej drogi do środka labiryntu. Wtedy możliwe jest skrócenie przeszukiwania, ponieważ znana jest już długość ścieżki, którą musi przebyć robot, żeby na pewno znaleźć się u celu. Nie ma więc sensu przeszukiwanie komórek o wadze większej. Pozostaje więc upewnienie się, czy wyznaczona trasa jest najkrótsza. Przeszukiwane są pozostałe, nieodwiedzone komórki. Aby do nich dojechać wykonywany jest algorytm Bellmana-Forda, przy czym za cel obierane jest pole, do którego robot ma trafić, a start stanowi pole, w którym aktualnie się znajduje. Po dojechaniu do wyznaczonej komórki następuje aktualizacja mapy i kolejna iteracja algorytmu Bellmana-Forda od początku labiryntu do jego środka. Mapowanie kończy się w momencie przeszukania całego labiryntu, tzn. wtedy kiedy zostały odwiedzone wszystkie komórki, których waga nie przekraczała wagi znalezionej ścieżki. Mając potrzebną mapę możliwe jest wyznaczenie optymalnej drogi.

Jak widać na rysunku 5.4d nie ma potrzeby sprawdzania pozostałych komórek, gdyż mają one wagę większą niż znaleziona najkrótsza ścieżka.

**Powrót** W celu dojechania do początku labiryntu wyszukiwana jest najkrótsza ścieżka od aktualnego położenia do komórki o numerze 0. W ten sposób robot przemieszczając się zgodnie z malejącymi wagami komórek może w jednoznaczny sposób powrócić na start. Zwrócić także trzeba uwagę na fakt, że trasa najkrótsza (czyli wymagająca przejechania najmniejszej liczby komórek do celu) nie zawsze będzie najszybsza. Wynika to z tego, że robot dłużej przemieszcza się o jedno pole, gdy musi wykonać skręt niż gdy może pojechać prosto. Aby wyznaczona trasa była najszybsza należy zmodyfikować tablice wag komórek zgodnie z tym w jaki sposób poruszać się będzie w labiryncie robot. Autor przyjął, że waga krawędzi gdzie nie potrzebna jest zmiana kierunku przyjmować będzie wartość 1, natomiast w przeciwnym przypadku waga ta będzie wynosić 2. Po stworzeniu kompletnej mapy oraz powrocie na start następuje modyfikacja wag krawędzi. Od tej chwili konstrukcja gotowa jest na przejazd czasowy (znana jest najszybsza trasa do celu).

10	11	10	9	8	9
9	10	9	8	7	8
8	11	0	0	6	9
7	12	0	0	5	10
6	13	14	1	4	11
5	4	3	2	3	12

(a) Labirynt przed zmianą ustalonych wag

12	14	13	12	10	13
11	13	12	11	9	11
10	15	0	0	8	13
9	16	0	0	7	14
8	17	19	1	6	15
6	5	4	2	4	16

(b) Labirynt po zmianie ustalonych wag

Rysunek 5.5: Zamiana wag i znalezienie najszybszej drogi- oznaczona na zielono (źródło: materiały autora)

**Przejazd czasowy** Analogicznie jak przy powrocie, jazda do celu odbywa się wybierając kolejne dostępne komórki o najmniejszych wagach. Ten przejazd może zostać wykonany z dużo większą prędkością, gdyż robot nie musi ani aktualizować mapy, ani wykonywać kolejnych iteracji algorytmu znajdowania najkrótszej ścieżki. Przejazd kończy się po dojechaniu do komórek o wadze 0.

## Rozdział 6

# Wnioski

Zgodnie z założeniami pracy opracowano oraz zaimplementowano rozwiązanie pozwalające na wyznaczanie pozycji robota w nieznanym środowisku labiryntu oraz znajdowanie najszybszej drogi z wykorzystaniem algorytmu Bellmana-Forda. Na podstawie doświadczeń z prowadzonych prac opracowano następujące wnioski:

- **Wykorzystanie EKF do wyznaczania pozycji**

Zastosowanie filtru pozwoliło na dokładniejszą estymację położenia robota, niż w przypadku wykorzystania wyłącznie pomiarów z czujników. Jednakże wraz z przebywaną przez konstrukcję odległością następuje powiększanie się błędu położenia. Wynika to z mechanizmu działania EKF (im dłuższy przebyty dystans, tym większe staje się prawdopodobieństwo, że błąd pozycji będzie narastał). Ponadto zjawisko to może być potęgowane nieefektywnym doбором wartości macierzy szumów procesowych  $\mathbf{Q}$  oraz pomiarowych  $\mathbf{R}$ . Opracowanie dokładniejszych metod wyznaczania odchyleń standardowych z pewnością poprawiłoby pracę filtru.

- **Czujniki odległości**

W przedstawionym rozwiązaniu czujniki odległości służą tylko i wyłącznie do wyznaczania obecności lub braku ścianek w danej komórce labiryntu. Jednakże mogłyby one posłużyć także do korekcy położenia robota poprzez pomiar odległości konstrukcji od znajdujących się obok niej ścianek. Takie podejście zniwelowałoby błąd położenia wynikający z działania filtru Kalmana.

Ponadto użyte czujniki w postaci diody IR i fototranzystora są czułe na zmiany oświetlenia otoczenia. Z tego względu droższym, lecz lepszym rozwiązaniem jest użycie czujników laserowych.

- **Interfejs użytkownika**

Ułatwieniem w modyfikacji oprogramowania robota jest wykorzystanie komunikacji bluetooth do przesyłania bieżących danych z konstrukcji do komputera i stworzenie wygodnego i przejrzystego interfejsu do prezentacji aktualnego stanu wykonywanego programu.

- **Mapowanie labiryntu**

Zastosowanie algorytmu Bellmana-Forda daje zadowalające wyniki zarówno w kroku mapowania, jak i przejazdu czasowego. W sprzyjających warunkach (szybkie dotarcie do środka

labiryntu w trakcie mapowania) pozwala na pominięcie przeszukiwania i przejazdu przez niektóre komórki labiryntu. Pozwala to na zaoszczędzenie czasu. Ponadto wyznaczona ścieżka jest zawsze najszybsza.

Ze względu na to że mikrokontroler ma ograniczone możliwości obliczeniowe oraz to, że w trakcie mapowania wykonywane są obliczenia zarówno algorytmu Bellmana-Forda, jak i Filtru Kalmana, przy każdej aktualizacji mapy robot musi się zatrzymać, aby dokonać odpowiedniej korekty ścieżki. Problem ten nie występuje przy przejeździe czasowym. Możliwym rozwiązaniem byłoby zastosowanie algorytmu wyznaczania najszybszej drogi o mniejszej złożoności obliczeniowej.

- **Przejazd przez labirynt**

W obecnej wersji robot może poruszać się po labiryncie w 4 kierunkach. Dużym usprawnieniem w działaniu konstrukcji byłaby zmiana algorytmu sterowania, aby umożliwić jazdę także na skos.

# Bibliografia

- [1] Encyklopedia algorytmów. *Algorytm Bellmana Forda*. 30 mar. 2017. URL: [http://algorytmy.ency.pl/arttykul/algorytm\\_bellmana\\_forda](http://algorytmy.ency.pl/arttykul/algorytm_bellmana_forda).
- [2] AMS. *AS5304 Linear Sensor Datasheet*. URL: <http://ams.com/eng/Products/Magnetic-Position-Sensors/Linear-Position/AS5304>.
- [3] Grewal; Andrews. *Kalman Filtering: Theory and Practice Using Matlaba*. John Wiley i Sons, 2001.
- [4] Karsten Berns. *Autonomous Mobile Robots*. Wykład zapraszany na Wydziale Mechanicznym Energetyki i Lotnictwa Politechniki Warszawskiej. 2016.
- [5] Bilgin Esme. *Kalman Filter For Dummies*. 16 mar. 2017. URL: <http://bilgin.esme.org/%20BitsAndBytes/KalmanFilterforDummies>.
- [6] Invensense. *MPU-6050 Datasheet*. URL: <https://www.invensense.com/wp-content/uploads/2015/02/MPU-6000-Datasheet1.pdf>.
- [7] Janusz Jakubiak. *Podstawy Robotyki, Wykład VII, Kinematyka robota mobilnego*. Wykład. 2015.
- [8] marcin13021988. *Roboty MicroMouse – 5 metod przeszukiwania labiryntu*. 20 wrz. 2016. URL: <http://forbot.pl/blog/arttykuly/programowanie/roboty-micromouse-5-metod-przeszukiwania-labiryntu-id17354>.
- [9] Maritex. *LIREDB-940 Datasheet*. URL: <https://botland.com.pl/diody-ir-podczerwone/3974-diody-lired3b-niebieska-940nm.html>.
- [10] Maritex. *LIRTB-940 Datasheet*. URL: <https://botland.com.pl/fototranzystory/1227-fototranzystor-lirt3b-940-940-nm.html>.
- [11] Wykłady MIMUW. *Zaawansowane algorytmy i struktury danych: Algorytm Bellmana-Forda*. 3 grud. 2017. URL: [http://wazniak.mimuw.edu.pl/index.php?title=Zaawansowane\\_algorytmy\\_i\\_struktury\\_danych/Wyk%C3%85%C2%82ad\\_5](http://wazniak.mimuw.edu.pl/index.php?title=Zaawansowane_algorytmy_i_struktury_danych/Wyk%C3%85%C2%82ad_5).
- [12] Fumitaka Nakashima. *My micromouse*. URL: <http://4th-laboratory.la.coocan.jp/mymouse.html>.
- [13] Koło Naukowe Robomatic. *Regulamin MicroMouse*. URL: [https://robomaticon.pl/files/pl/rules/Regulamin\\_MicroMouse.pdf](https://robomaticon.pl/files/pl/rules/Regulamin_MicroMouse.pdf).
- [14] Illah Reza Nourbakhsh Roland Siegwart. *Introduction to Autonomous Mobile Robots*. 2004. DOI: <http://home.deib.polimi.it/gini/robot/docs/siegwart.pdf>.
- [15] Stan. *Differential steering with continuous rotation servos and arduino*. 3 13 2014. URL: <http://42bots.com/tutorials/differential-steering-with-continuous-rotation-servos-and-arduino/>.

- [16] Cormen Leiserson Rivest Stein. *Wprowadzenie do algorytmów*. Wydawnictwo Naukowe PWN, Warszawa, 2012.
- [17] STMicroelectronics. *STMF103RBT6 Datasheet*. URL: [www.st.com/resource/en/datasheet/CD00161566.pdf](http://www.st.com/resource/en/datasheet/CD00161566.pdf).
- [18] Gabriel A. Terejanu. *Extended Kalman Filter Tutorial*. Artykuł. URL: <https://homes.cs.washington.edu/~todorov/courses/cseP590/readings/tutorialEKF.pdf>.
- [19] Polska Unia Robotyki Turniejowej. *Regulamin Micromouse*. URL: [http://www.robomotion.rzeszow.pl/Regulaminy\\_PURT\\_micromouse.pdf](http://www.robomotion.rzeszow.pl/Regulaminy_PURT_micromouse.pdf).
- [20] Jerzy Wałaszek. *Algorytmy i struktury danych: Reprezentacja grafów w komputerze*. Artykuł. 2014. URL: [http://eduinf.waw.pl/inf/alg/001\\_search/0124.php](http://eduinf.waw.pl/inf/alg/001_search/0124.php).
- [21] Green Ye. *Green Ye's personal project site*. URL: <http://www.greenye.net/>.

# Spis rysunków

1.1	Labirynt konkurencji Micromouse (źródło: [8]) . . . . .	12
1.2	Robot Green Giant V4.2 (źródło: [21]) . . . . .	13
1.3	Robot Yukikaze (źródło: [12]) . . . . .	14
2.1	Robot Ariadna wykorzystany w pracy-model CAD (źródło: materiały autora) . . . . .	15
3.1	Schemat sterowania robotem dwukołowym o napędzie różnicowym ( od lewej: jazda na wprost, skręt po łuku w lewo, skręt w miejscu w lewo) (źródło: [15]) . . . . .	22
3.2	Więzy kinematyczne napędu różnicowego (źródło: [7]) . . . . .	23
3.3	Sprowadzanie układu różnicowego do modelu monocykla (źródło: [7]) . . . . .	24
3.4	Pozycja robota w obranym układzie odniesienia (źródło: materiały autora) . . . . .	25
5.1	Numeracja pól labiryntu (źródło: materiały autora) . . . . .	40
5.2	Schemat labiryntu, w którym nie działa metoda prawej strony (źródło: [8]) . . . . .	43
5.3	Przykład reprezentacji labiryntu za pomocą grafu (źródło: materiały autora) . . .	46
5.4	Przykładowy przejazd mapujący labirynt 6x6 (czerwone krawędzie oznaczają nieodkryte ściany, żółte pola nieodwiedzone komórki) (źródło: materiały autora) . .	48
5.5	Zamiana wag i znalezienie najszybszej drogi- oznaczona na zielono (źródło: materiały autora) . . . . .	49
A.1	Schemat elektroniczny mikrokontrolera . . . . .	57
A.2	Schemat elektroniczny peryferiów . . . . .	58

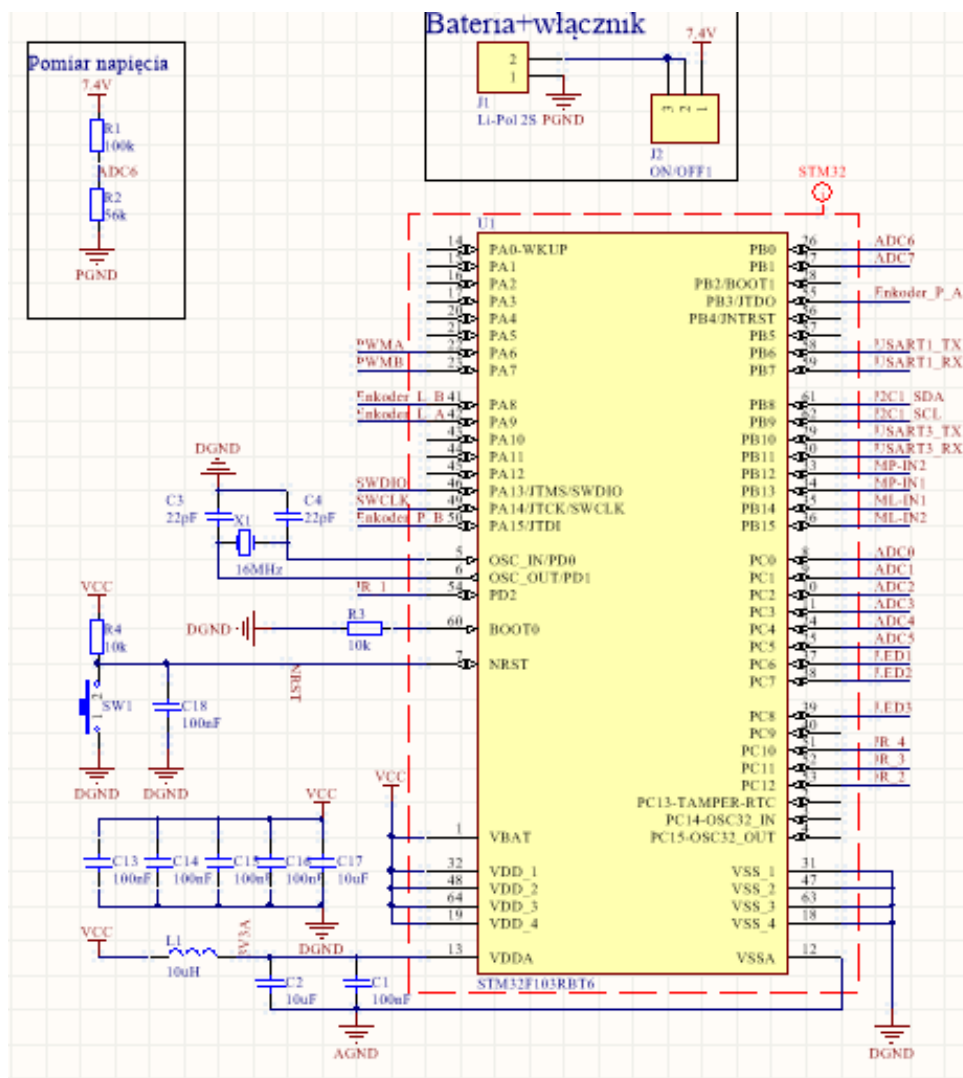
# Spis tablic

1.1	Charakterystyka robota Green Giant V4.2(źródło: [21]) . . . . .	13
1.2	Charakterystyka robota Yukikaze (źródło: [12]) . . . . .	14
2.1	Charakterystyka robota Ariadna (źródło: materiały autora) . . . . .	16
4.1	Pomiary orientacji robota . . . . .	38
4.2	Pomiary przejazdu po prostej . . . . .	39

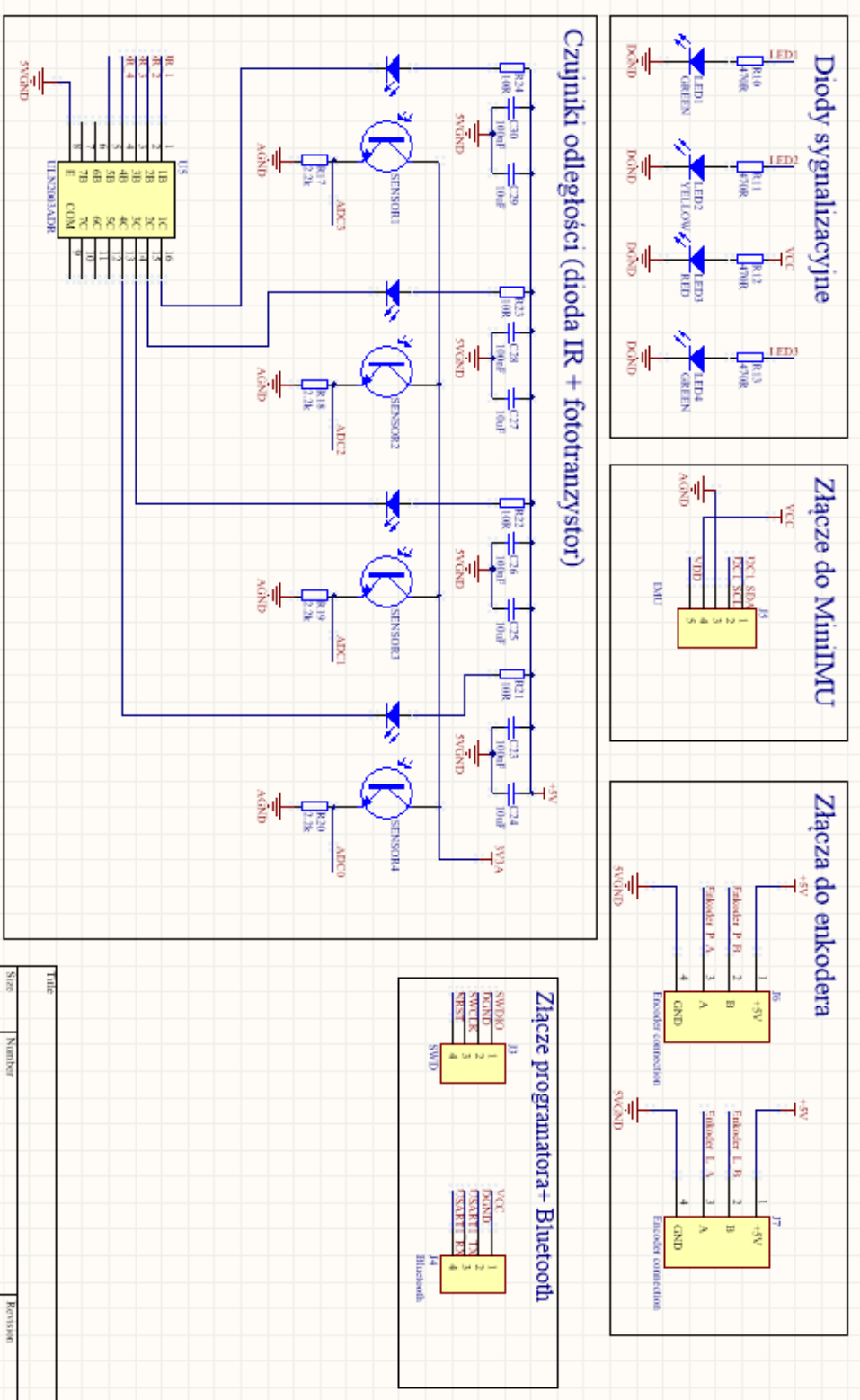


## Dodatek A

# Załącznik: Schematy elektroniczne robota Ariadna



Rysunek A.1: Schemat elektroniczny mikrokontrolera



Rysunek A.2: Schemat elektroniczny peryferiów