

Съфинансиран от програма  
„Еразъм+“  
на Европейския съюз



## МЕХАТРОНИКА И РОБОТИКА ЗА СРЕДНИТЕ УЧИЛИЩА

### ВЪВЕДЕНИЕ В РОБОТИКАТА, МЕХАТРОНИКАТА И ПРОГРАМИРАНЕТО



Creative Commons licence  
Attribution-NonCommercial-  
ShareAlike CC BY-NC-SA



Проект “Developing Innovative  
Science Outreach for Vocational  
Education to Encourage STEM  
Careers and Education”, №2017-1-  
BG01-KA202-036327



Дата на публикуване: 2019

## Съдържание

<b>Въведение</b>	<b>3</b>
Цели на обучението	
Редактори	
<b>Основни системи и компоненти, необходими за конструирането на робот</b>	<b>4</b>
Основни системи и компоненти	
Оценка на знанията	
<b>Какво представлява програмирането</b>	<b>7</b>
Каква е ролята на програмата в робота	
Структура на програмата	
За какво служат променливите и константите в програмите	
Използване на подпрограми и функции	
Блок прекъсвания в програмата	
Оценка на знанията	
<b>Вземане на решения в програмите</b>	<b>14</b>
Логически операции в програмите	
Разклоняващи структури в програмите с оператори if ... Else	
Разклоняващи структури с оператора за безусловен преход goto	
Оценка на знанията	
<b>Цикли в управляващите програми</b>	<b>19</b>
Какво налага необходимостта от използването на цикли	
Как да създаваме на цикли	
Оценка на знанията	
<b>Движения и позициониране на робота</b>	<b>24</b>
Характеристики на мобилните роботи	
Управление на двигателите	
<b>Сензори в роботите</b>	<b>29</b>
Зашто роботите имат нужда от сензори?	
Сервомотор. Сензор за въртене на двигателя (енкодер)	
Сензор за допир	
Сензор за разстояние	
Сензор за светлина	
Сензор за цвят	
Сензор за звук	
Сензор за ускорение (акселерометър)	
Сензор-компас	
Жироскопичен сензор	
Инфрачервен сензор	
Сензор за откриване на обекти	
Оценка на знанията	
<b>Анекс 1: Допълнителна информация</b>	<b>40</b>
Роботи	40
Алгоритъм в програмирането	45
Разклоняващи структури в програмите с оператори switch .. case	49
Видове програмни езици	51
Изпълнителни устройства и задвижвания	52
Физически характеристики на механичното движение	58
Сензорът като чувствителен елемент в системата на робота	64

Следене на линия	66
Избягване на препятствия	87
Комуникация с други работи или персонален компютър	95



## Въведение

### ЦЕЛИ НА ОБУЧЕНИЕТО

Последното десетилетие значително се увеличи интересът към използването на роботите в образованието. Уникалността на образователната роботика се състои във възможността да се обедини конструирането и програмирането, което способства за интегриране на информатиката, математиката, физиката и естествените науки с развитие на инженерното мислене. Тя позволява да се запознаят учениците с основите на алгоритмизацията, построяването на комплексна система, развитие на конструкторско мислене и навици за решаване на сложни задачи.

Целта на предлагания учебен материал е да подпомогне учениците от средните училища да развият своите практически знания и умения в областта на мехатрониката, роботиката и програмирането. Това ще ги мотивира да продължат образованието и кариерното си развитие в областта на технологиите и инженерството.

Учебният материал има за цел да развие практическите знания и умения на учениците при конструирането на роботи и тяхното програмиране. Той е базиран на робота Lego Mindstorms EV3 и програмирането му на езика С. Работата с конструктора LEGO Mindstorms EV3 дава възможност на учениците в игрова форма да усвоят основите на програмирането на роботизирани устройства, което в бъдеще може да бъде екстраполирано към комплексни задачи и проекти. Разработените уроци могат да дадат на учениците основи в разбиранията за програмиране и робототехника. В тях се акцентира на особеностите при написване на програма, движението и позиционирането на робота, сензорите в робота, следене на линия, избягване на препятствия, комуникацията на робота с други роботи и персонален компютър.

### РЕДАКТОРИ

Станимир Йорданов и Тодор Тодоров, Технически университет - Габрово.

## Основни системи и компоненти, необходими за конструирането на робот

### ОСНОВНИ СИСТЕМИ И КОМПОНЕНТИ

Работът е устройство, което се състои от различни модули и системи. Основните от тях са:

- сензорна система;
- система за задвижване;
- система за управление;
- система за захранване.

Сензорната система се състои от различни типове сензори, чрез които роботът получава информация от обкръжаващата го среда. Сензорите могат да бъдат за температура, влажност, налягане, светлина, допир, енкодер, акселерометър, ултразвуков сензор, камера и др.

Поради това, че по-нататъшното разглеждане на роботите и тяхното програмиране в настоящия учебен материал е ориентирано към мобилния робот, предлаган от фирмата LEGO - *LEGO MINDSTORMS® EV3, се представят компонентите от отделните системи на този робот.*

Към комплекта на *LEGO MINDSTORMS® EV3* са включени следните типове сензори:

- сензор за светлина от видимия и инфрачервения спектър;
- фоторастерен сензор за движение – енкодер;
- акселерометър;
- сензор за допир;
- ултразвуков сензор;
- жироскоп;
- сензор компас и др.

Другата основна система на всеки робот е системата за задвижване. Чрез нея се реализират всички движения, извършвани от робота. Задвижването се осъществява от някакъв тип двигател. Към комплекта на *LEGO MINDSTORMS® EV3* са предоставени следните задвижващи модули:

- серводвигатели;
- за линейно преместване;
- за повишаване на мощността, повишаване или понижаване на обороти на двигател.

За по-подробно описание на сензорите и двигателите, вижте Анекс 1: Допълнителни материали.

Системата за управление обединява всички останали системи на

робота и управлява неговите действия. Чрез нея се събира информация от сензорите, която се обработва и като резултат се управлява системата за задвижване на робота. Тя се реализира с помощта на микроконтролер.

Системата за управление на *LEGO MINDSTORMS® EV3* е изградена на базата на микроконтролер TI Sitara AM1808 с ядро ARM926EJ-S 300MHz, монохромен LCD дисплей с резолюция 178×128 пиксела, оперативна памет 64 MB RAM за изпълнение на програмата и памет за съхранение на програми и данни 16 MB Flash. Тя е снабдена с microSDHC Slot, към който може да бъде добавена допълнителна памет. На фигураната по-долу е показан управляващият модул на *LEGO MINDSTORMS® EV3*.



*Модул за управление на LEGO MINDSTORMS® EV3*

Модулът за захранване на *LEGO MINDSTORMS® EV3* е реализиран с вградена li-ion батерия. За зареждане на батерията е предоставен адаптер.

Към комплекта на *LEGO MINDSTORMS® EV3* е предоставен набор от различни конструктивни елементи, с помощта на които може да се изгради желаното механично устройство (виж фигураната по-долу).



*Комплект елементи за изграждане на мобилен робот*

Подкрепата на Европейската комисия за изготвянето на настоящата публикация не представлява одобрение на съдържанието, което отразява гледните точки само на авторите и не може да се търси отговорност от Комисията за всяка употреба, която може да бъде използвана за информацията, съдържаща се в нея.

**ОЦЕНКА НА  
ЗНАНИЯТА**

1. Кои от изброените компоненти не се използват в сензорната система на робота?

- а) сензор за температура
- б) електродвигател
- в) акселерометър
- г) хидравличен цилиндър
- д) реле

2. С какво се отчита ъгълът на завъртане на електродвигателя?

- а) сензор за налягане
- б) ултразвуков сензор
- в) фоторастерен преобразувател – енкодер
- г) сензор за разстояние
- д) модул за GPS координати

3. С каква честота работи микроконтролерът на LEGO MINDSTORMS® EV3?

- а) 3000MHz
- б) 300MHz
- в) 200MHz
- г) 500MHz
- д) 1300MHz

*Верни отговори: 1 (б, г, д); 2(в); 3(б)*



## Какво представлява програмирането

### КАКВА Е РОЛЯТА НА ПРОГРАМАТА В РОБОТА

Управлението на робота се извършва от микроконтролер, на базата на който се реализира системата му за управление. За да изпълнява управляващите функции, микроконтролерът се нуждае от програма. Програмата представлява набор от инструкции, които се изпълняват по определен ред и се реализира съответен алгоритъм. В резултат от изпълнението на програмата се обработва информацията от сензорите и се управлява системата за задвижване на робота. Програмата е тази, която прави от робота едно цялостно завършено устройство, което може да изпълнява определени функции.

### СТРУКТУРА НА ПРОГРАМАТА

Като цяло програмата се пише и редактира най-често с текстов редактор. Инструкциите, които се използват са в пряка зависимост от езика, който се използва за програмиране на съответния микроконтролер. Когато бъде написана дадена програма, тя трябва да се компилира до машинен код за съответния микроконтролер. Това се извършва с компилираща програма. След това компилираният код се записва в микроконтролера. Обикновено микроконтролерите се програмират със специален хардуер, наречен програматор. В последно време се предлагат микроконтролери, при които програмирането се извършва директно през USB или USART порта на компютъра. След като компилираният код се запише в микроконтролера, трябва да се извърши тестване на системата. За тестването към микроконтролера и цялата система се подава необходимото захранване и се стартира неговата работа. По време на изпълнение на програмата се следи работата на микроконтролера и как той управлява наличната му периферия. Тъй като няма как да погледнем какво става вътре в микроконтролера, се наблюдават действията на робота и се следи дали те реализират това, което е заложено в алгоритъма и програмата.

Обикновено структурата на програмата се състои от една главна функция, в която се описват всички действия, които тя трябва да извърши. За да се изпълни кодът на програмата, тя трябва да се стартира, като операционната система ѝ предава управлението. След завършване на работата на програмата, управлението се връща отново на операционната система. Така работят компютрите.

При микроконтролерите се използва друг подход. При включване на захранването се стартира програмата, която е записана в него. Тази програма съдържа в себе си четири основни области или полета, в които се записват всички команди, необходими за правилната работа на системата.

В първата област се поставят командите, с които се дефинират всички библиотеки, съдържащи функциите, които ще се използват в програмата. В това поле се дефинират всички константи, макроси и се обявяват променливите, които се явяват

*Подкрепата на Европейската комисия за изготвянето на настоящата публикация не представлява одобрение на съдържанието, което отразява гледните точки само на авторите и не може да се търси отговорност от Комисията за всяка употреба, която може да бъде използвана за информацията, съдържаща се в нея.*

като главни за програмата. Също така тук се обявяват всички функции, които ще се създават за ползване от програмата и се намират в нея. Тези функции ще бъдат описани в последното четвърто поле. На фиг. 1 е показана първата област (полето) от програмата, с описание на библиотеки, макроси, константи, глобални променливи и функции.

```

1 #include <AIUT_pins_NANO.h>           // including a header file
2 #include <SoftwareSerial.h>           // including a header file
3
4 #define key while(getchar() != '\n') // defining a macro
5 #define pi(r) 2*pi*r;              // defining a macro
6 #define pin2 = 2;                  // defining a constant
7 #define pin3 = 3;                  // defining a constant
8 #define pin5 = 5;                  // defining a constant
9 #define PI 3.1415926             // defining a constant
10 const float pi = 3.1415926;        // defining a constant
11 float Per, per;                 // defining a global variable of type float
12 String inputString; // defining a global variable of type String
13 bool stringComplete;            // defining a global variable of type bool
14
15 void setOput(byte chanel, char state); // function declaration
16 int readTemperature(int chanel);       // function declaration
17 byte getInput(byte chanel);           // function declaration
18

```

*Фиг. 1: Първо поле с описани библиотеки, макроси, константи, глобални променливи и функции на програмата*

Втората област е организирана във функция за инициализация на системата. В нея се описват всички операции по инициализиране и настройване на контролера, които са необходими за правилното му функциониране. Тази функция се изпълнява само веднъж при първоначално стартиране на програмата или на контролера.

На фиг. 2 е показана втората област от програмата, с описание на функцията за инициализация на контролера.

```

20 void setup() {
21   pinMode(pin2, OUTPUT);    // initializing a pin as output
22   pinMode(pin3, OUTPUT);    // initializing a pin as output
23   pinMode(pin5, INPUT);     // initializing a pin as input
24   Serial.begin(9600);      // calling a function from a library
25   inputString.reserve(200);
26   Serial.println("Arduino NANO of AIUT\n");
27   inputString = "";
28   stringComplete = false;
29 }
30

```

*Фиг. 2: Функция за инициализиране на микроконтролера*

Третата област от програмата е организирана като главна функция на цялата програма. В нея се описват всички операции и команди, с помощта на които се реализира желаният алгоритъм от действия на цялата програма. Функцията е организирана като цикъл така, че при достигане на нейния край, тя започва изпълнението си отново. Това се повтаря до изключване на захранването на контролера или до натискане на бутона reset и рестартиране на системата.

На фиг. 3 е показана третата област от програмата, организирана като главна функция на програмата.



```

31 void loop() {
32     int i, j; // defining a local variable of type int
33     float x, y, z; // defining a local variable of type float
34     byte a, b; // defining a local variable of type byte
35     a = getInput(pin5); // calling a function in the program
36     if(a == true){
37         setOutput(pin2, "1");// calling a function in the program
38     }
39 }
40
41

```

Фиг. 3: Главна функция на програмата на микроконтролера

В последната четвърта област от програмата се описва подробно съдържанието и начинът на действие на всички вътрешни функции за програмата. За всяка функция се описват броя и типа на параметрите, които трябва да ѝ се подадат при нейното извикване. Също така се обявява и типа на данните, които функцията ще върне към главната програма при своето завършване.

На фиг. 4 е показана четвъртата област, където се описват всички вътрешни функции на програмата.

```

42 void setOutput(byte chanel, char state) {
43     if(state == '0')
44         digitalWrite(chanel, LOW);// calling a function from a library
45     else
46         digitalWrite(chanel, HIGH);// calling a function from a library
47     }
48
49 -----
50 byte getInput(byte chanel) {
51     bool r; // defining a local variable for the function
52     r = digitalRead(chanel );// calling a function from a library
53     return r; // returning value to the main program
54 }

```

Фиг. 4: Четвърта област с описание на вътрешните функции, използвани от програмата на микроконтролера

## ЗА КАКВО СЛУЖАТ ПРОМЕНЛИВИТЕ И КОНСТАНТИТЕ В ПРОГРАМИТЕ

Променливите и константите са основна част от текущите данни, благодарение на които е възможна работата на програмата в контролера. За целта в RAM паметта на микроконтролера се запазва специално място, където се извършва съхранението на тези данни. За да може да се работи с константи и променливи, при създаването им се описва и името, с което ще се асоциира дадена променлива, както и съдържащата се в нея стойност.

Константите съдържат в себе си стойности, които остават непроменени през цялото време докато програмата работи.

За разлика от константите променливите съхраняват в себе си стойности, които са моментни. Тези стойности могат да бъдат променяни по всяко време от операторите и командите на програмата. За да се вземе съдържанието на една променлива се използва нейното име. Името на променливите се задава свободно от програмиста като се спазват правилата за тях. Имената трябва да са изписани на латиница и в зависимост от компилатора могат да съдържат в себе си цифри и специални символи. Най-често имената са описателни и са свързани с това, за което ще се

*Подкрепата на Европейската комисия за изготвянето на настоящата публикация не представлява одобрение на съдържанието, което отразява гледните точки само на авторите и не може да се търси отговорност от Комисията за всяка употреба, която може да бъде използвана за информацията, съдържаща се в нея.*



използва дадената променлива. Чрез името ѝ променливата може да се впише във формула, да се предава като параметър, да се отпечатва съдържанието ѝ и т.н.

Специфичното при променливите е, че те могат да бъдат от различен тип. Типът на променливата се задава при самото ѝ създаване и не може да бъде променяно. Според типа на променливата се запазва определено количество (брой клетки) памет. За всички променливи, които са от един и същ тип, се запазва еднакъв брой клетки от паметта. Типовете данни за променливи биват:

- **Цели числа** – цели числа (1, 2, 3, ..., 102, ..., 1003, и т.н.);
- **Реални числа** – дробни числа (0.1, 2.5, 3.8, ..., 10.2, ..., 100.3, и т.н.);
- **Символи** – символи (a, b, D, E, и т.н.);
- **Стринг** – поредица от символи или текст (Това е текст!, и т.н.);
- **Указатели** – променливи, които съдържат в себе си адрес, показващ къде се намира стойността или съдържанието;
- **Булеви** – променливи с две състояния - true и false.

Синтаксисът за деклариране на променливи в C++ и в повечето езици е следния:

**<тип данни> <идентификатор или име> [= <инициализация или стойност>].**

Пример за деклариране на променливи може да се види на редове 11, 12, 13 от фиг. 1, на редове 33, 34 от фиг. 3 и на ред 51 от фиг. 4.

Зоната на действие, видимостта и времето на живот на променливата, зависи от това къде тя е декларирана. Ако променливата е декларирана както е показано на редове 11, 12, 13 от фиг. 1, то тя е глобална. Променливата, декларирана на ред 51 от фиг. 4 е локална.

Глобалните променливи са видими от цялата програма, а също така и от вътрешните функции на програмата. Времето им на живот е до спиране на програмата.

Локалните променливи са видими само във функцията, в която са деклариирани.

Променливите, деклариирани в главната програма на редове 33, 34 от фиг. 3 имат глобален и локален характер едновременно. За главната програма те се явяват глобални, но за функциите, описани извън главната програма, те остават невидими и недостижими. Стойностите на такива променливи се подават към функциите най-често като параметър при извикването на

## ИЗПОЛЗВАНЕ НА ПОДПРОГРАМИ И ФУНКЦИИ

функцията.

Много често при писане на програми, се достига до ситуация, при която един и същи код се налага да се пише по няколко пъти. За да се избегне това, този повтарящ се код може да се организира като отделна функция или подпрограма. Изпълнявайки главната програма, на местата където е необходимо да се получи резултатът от съответното изчисление се извиква тази функция и управлението се предава на нея. След завършване на своите действия, функцията отново предава управлението на главната програма, като връща и резултата от своите изчисления.

Синтаксисът за деклариране на функция в повечето езици е следния:

**<връщан тип> <име> ([тип] [параметър1], [тип]  
[параметър2],...).**

На фиг. 5 е показана структурата и кода на примерна функция.

```

29
30 int CalculateOper(int operand1, int operand2) {
31     int x;
32     x = operand1 * operand2;
33     return x;
34 }
```

*Фиг. 5: Примерен код на функция*

Наличието на запазената дума **int** означава, че функцията ще върне някакъв резултат от своите действия. В случая това ще бъде цяло число. На ред 32 от фиг. 5 се извършва действието (в случая умножение) и резултатът се приема от променливата **x**. Операторът **return** връща управлението на главната програма като предава и стойността на **x**.

Извикването на функцията от главната програма е показан на фигурата по-долу.

```

15 void loop() {
16     int Resultat;
17
18     Resultat = CalculateOper(4, 5);
19
20 }
```

*Фиг. 6: Извикване на функция*

При извикването на функцията, към нея трябва да се подадат необходимите параметри. В случая тук има два параметъра, които са поставени в скобите на ред 18 от фиг. 6. Броят на предаваните параметри може да бъде толкова, колкото е необходимо за работа на функцията. Типът и броят на параметрите се задават от програмиста, при първоначалното създаване на функцията и се декларират в началото на програмата. Резултатът, който се връща от функцията, се предава на променлива от същия тип, от който е обявена функцията. В случая това е променливата **Resultat**, ред 18 от фиг. 6. Променливата трябва да е обявена като тип **int**, както се



вижда на ред 16 от фиг. 6.

Функцията може да бъде организирана така, че да не връща никакъв резултат. Това в някои езици е определено като процедура. Например, в езика **Pascal** има процедури и функции, докато в езикът **C++** има само функции.

Чрез използването на функции и процедури, кодът на програмата става много по-четим, кратък и ясен.

## БЛОК ПРЕКЪСВАНИЯ В ПРОГРАМАТА

Прекъсванията са процедури, които са изключително полезни и повишават производителността на всеки един микроконтролер. Без тях не би било възможно да се реализира работа на системите, които се нуждаят от така нареченото управление в реално време. Както и самото им име подсказва, че това е прекъсване на главната програма на микроконтролера, предизвикано от някакво събитие, след което управлението се предава на друг процес (подпрограма, процедура).

Във всеки микроконтролер има заложен механизъм на прекъсване, който е наречен блок за обработка на прекъсване. Този механизъм може да бъде реализиран софтуерно (изпълнение на програмен код) или хардуерно (хардуерна схема, изпълняваща определена поредица от действия). Прекъсванията са процес, който може да се предизвика по два начина - софтуерно и хардуерно. Другото име, под което се срещат в литературата, е вътрешно или външно прекъсване за микроконтролера. Вътрешните прекъсвания за микроконтролера обикновено се изработват от различните видове таймери, които са в микроконтролера. Външни за микроконтролера са прекъсванията, които се предизвикват от подаване на външен сигнал (с високо или ниско ниво) на определени входове на микроконтролера. Всички прекъсвания (софтуерни или вътрешни и хардуерни или външни) се делят на такива с висок приоритет и с нисък приоритет. Ако е започнала обработка на прекъсване с нисък приоритет и в същото време се получи заявка за прекъсване с висок приоритет, то тогава се прекъсва изпълнението по обслужване на ниския приоритет и се извършва изпълнение на прекъсването с висок приоритет. След завършване на обслужването на прекъсване с висок приоритет, се продължава с обслужването на прекъсването с нисък приоритет, от там до където е било достигнalo преди това. Програмистът трябва да реши, към какъв приоритет да насочи обработката на събитията, които се случват при работата на програмата.

Процедурата, която се изпълнява при прекъсване, е желателно да бъде възможно най-кратка. Обикновено това е промяна на стойността на една или няколко променливи. Най-често тези променливи са от тип **bool** и се наричат флагове. След края на процедурата, управлението отново се връща към главната програма. Според алгоритъма, който изпълнява, главната програма следи състоянието на тези флагове. В зависимост от промяната на флаговете (вдигнат флаг, когато имаме логическа единица), се

*Подкрепата на Европейската комисия за изготвянето на настоящата публикация не представлява одобрение на съдържанието, което отразява гледните точки само на авторите и не може да се търси отговорност от Комисията за всяка употреба, която може да бъде използвана за информацията, съдържаща се в нея.*



**ОЦЕНКА НА  
ЗНАНИЯТА**

изпълняват и съответните подпрограми, които са необходими за да се изработи желаното действие от системата. При достигане на края на тези подпрограми, състоянието на флага се нулира, което е знак за програмата, че събитието е обработено.

1. Кои са начините за предизвикване на прекъсване в микроконтролера?

- a) паралелно
- б) разклонено
- в) софтуерно
- г) циклично
- д) хардуерно

2. Какви биват прекъсванията спрямо начина им на обслужване?

- а) циклични
- б) разклонени
- в) с висок приоритет
- г) с нисък приоритет
- д) със среден приоритет
- е) паралелни

3. С кой тип трябва да се обявят променливите, които ще се използват за съхранението на цели числа?

- а) bool
- б) real
- в) int
- г) float
- д) char

*Верни отговори: 1 (б, д); 2 (б, з); 3(б)*



## Вземане на решения в програмите

### ЛОГИЧЕСКИ ОПЕРАЦИИ В ПРОГРАМИТЕ

Логическите операции са в основата за вземането на решение в алгоритмите и програмите. Най-общо казано логическите операции са действия, които осигуряват резултат (истина или лъжа), при различни сравнения между стойностите на две променливи или променлива и константа. Синтаксисът на логическите изрази е следния:

**<параметър 1> < логически израз > <параметър 2>.**

На мястото на *параметър 1* и *параметър 2*, се поставят стойностите от две променливи или от променлива и константа. Логическият израз може да бъде един от показаните на фиг. 7.

```

70 == Are the values of the two parameters equal?
71 != Are the values of the two parameters different?
72 < Is the value of the first parameter less than the second?
73 <= Is the value of the first parameter less than or equal to the
74 second?
75 > Is the value of the first parameter greater than the second?
76 >= Is the value of the first parameter greater than or equal to the
77 second?
&& Are both parameters true?
|| Is there at least one parameter that is true?

```

Фиг. 7: Логически изрази

Ако може да се отговори с ДА на всеки един от въпросите за логическите изрази от фиг. 7, то тогава резултатът от условието (сравнението) е **true** (истина). В противен случай, ако отговорът е НЕ, то тогава резултатът от условието е **false** (лъжа).

### РАЗКЛОНИВАЩ И СТРУКТУРИ В ПРОГРАМИТЕ С ОПЕРАТОРИ IF ... ELSE

За вземането на някакво решение при програмите се използват специфични оператори. Най-елементарният от тях е операторът **if**. Чрез него програмата може да провери дадено условие (параметър, променлива) и да изпълни един или друг набор (блок) от команди. Има няколко вида разклонения.

Първият вариант е само с оператора **if** при изпълнението на логическия израз (сравнение) в дадено условие:

- Ако резултатът от сравнението е **true**, то тогава се изпълнява допълнителният програмен код, който се намира в блока (заграден с големи скоби) на оператора **if** и след това програмата продължава с кода, който е след блока.
- Ако резултатът от сравнението е **false**, то тогава този допълнителен програмен код не се изпълнява, а програмата продължава с кода, който е след блока на оператора **if**.

На фиг. 8 е показан синтаксисът на този вариант на разклонение.

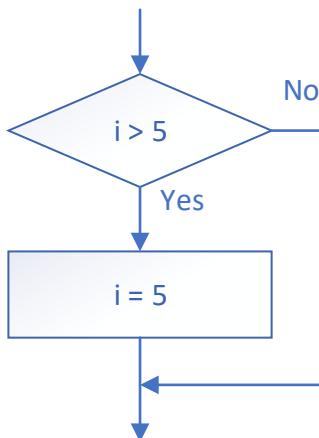
```

69 if(condition) {
70   operator 1
71   operator 2
72   ...
73   ...
74   operator n
75 }

```

Фиг. 8: Синтаксис на оператора if

На фиг. 9 е показана блокова схема на алгоритъм за вариант на разклонение с оператора **if**.



Фиг. 9: Блокова схема на алгоритъм за разклонение с оператора if

Вторият вариант е с оператора **if ... else**. Това буквально се чете или едното или другото. В зависимост от резултата от сравнението в логическия израз, се изпълнява един или друг програмен код:

- Ако резултатът от сравнението е **true**, то тогава се изпълнява първият програмен код, който се намира в блока (заграден с големи скоби) на оператора **if** от ред 93 до 97. След това се прескача кодът, който е в блока на **else** и програмата продължава с кода на основната програма, който е след ред 104 от фиг. 10.
- Ако резултатът от сравнението е **false**, то тогава се прескача кодът от **true**, от ред 93 до 97. Изпълнява се кодът от блока **else**, от ред 99 до 103 и програмата продължава с кода, който е след блока на оператора **else** (ред 104).

На фиг. 10 е показан синтаксисът на този вариант на разклонение **if ... else**.

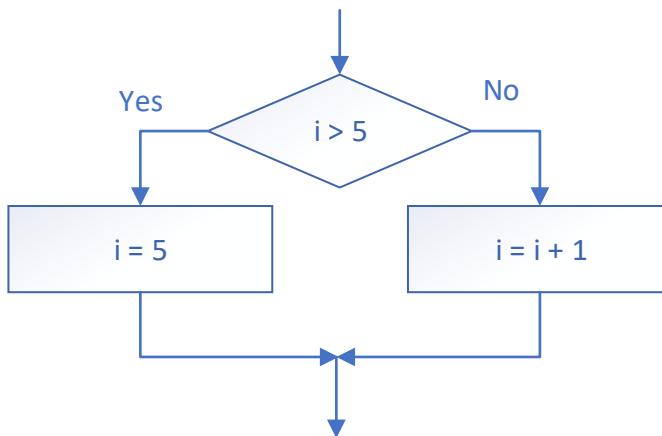
```

92 if ( condition ) {
93   operator 1
94   operator 2
95   ...
96   ...
97   operator n
98 } else {
99   operator 1
100  operator 2
101  ...
102  ...
103  operator n
104 }

```

Фиг. 10: Синтаксис на оператора if .. else

На фиг. 11 е показана блокова схема на алгоритъма за вариант на разклонение с оператора **if... else**.



Фиг. 11: Блокова схема на алгоритъм за разклонение с оператора if .. else

Третият вариант е с вложени оператори **if... else**. Много често при по-сложни логически зависимости се налага използването на вложени оператори **if ... else**. В зависимост от резултата от сравнението във всеки логически израз, се изпълнява един или друг програмен код, като се прилагат указанията описани по-горе. Няма ограничение за броя на използвани вложени оператори.

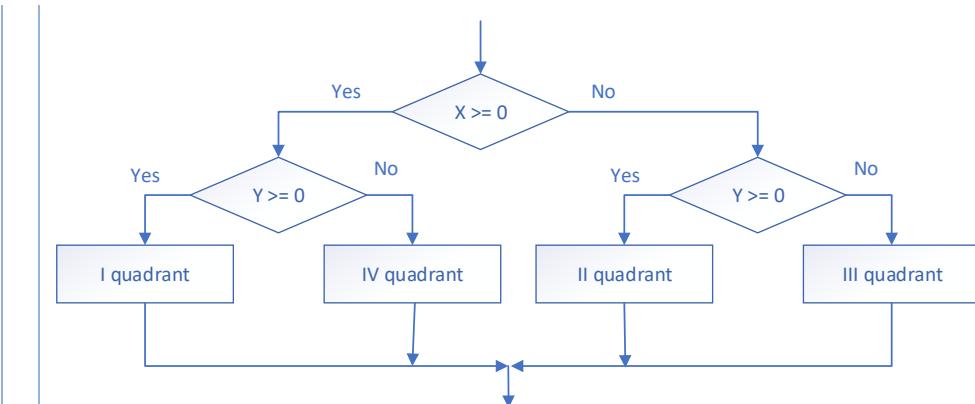
На фиг. 12 е показан синтаксисът на този вариант на разклонение с вложени оператори **if... else**.

```

108  if ( condition ){
109    if (condition ){
110      ...
111      ...
112    }else{
113      ...
114      ...
115  }
116  } else {
117    if (condition ){
118      ...
119      ...
120    }else{
121      ...
122      ...
123  }
124  ...
125  ...
126 }
```

Фиг.12: Синтаксис на вложени оператори if .. else

На фиг. 13 е показана блокова схема на алгоритъм за разклонение с вложени оператори **if.. else**.



Фиг. 13: Блокова схема на алгоритъм за разклонение с вложени оператори if .. else

### РАЗКЛОНИВАЩИ СТРУКТУРИ С ОПЕРАТОРА ЗА БЕЗУСЛОВЕН ПРЕХОД *GOTO*

Понякога макар и рядко се налагат изключения, поради което логиката на алгоритъма води до това, че не може да бъде описана с представените до сега оператори. Това налага използването на оператора за безусловен преход *goto*. Той безусловно препраща управлението на програмата от ред, на който е поставен оператора *label*. Операторът се съчетава с цифра, тъй като трябва да бъде уникатен и неповторим. Възможно е преходът към съответния етикет *label* да бъде извършен от няколко оператора *goto*, намиращи се на различни места в кода на програмата. Не се допуска обаче да има повече от един оператор *label* с еднакви номера.

На фиг. 14 е показан синтаксисът на оператора за безусловен преход *goto* и *label*.

При достигане на програмата от фиг. 14 до ред 161, се изпълнява операторът за безусловен преход и управлението на програмата се предава на кода от ред 156, от където продължава надолу. При използването на този вид оператори трябва да се внимава програмата да не влезе в безкраен цикъл. Излизането от него е невъзможно и се извършва само с рестартиране на микроконтролера.

```

136 void loop() {
137     byte a, b; // defining a local variable of type byte
138     a = getInput(pin5); // calling a function in the program
139     ...
140     label 1;
141     ...
142
143     if(a == true){
144         setOutput(pin2, "1"); // calling a function in the program
145         goto label 1;
146     }
147 }
148
  
```

Фиг. 14: Синтаксис на оператора за безусловен преход *goto*

**ОЦЕНКА НА  
ЗНАНИЯТА**

1. Кои от операциите се използват за логически изрази?
    - a) ==
    - б) конкатенация
    - в) &&
    - г) --
    - д) ++
    - е) >+
  2. Кои запазени думи участват в структурата на разклонение?
    - a) while
    - б) if ... else
    - в) goto
    - г) else ... if
    - д) for
  3. При коя от операциите може да се изпълни единият от два различни блока с команди?
    - a) else ... while
    - б) if ... label
    - в) if ... else
    - г) do ... while
  4. В структурата **if(условие) else**, кой блок с команди се изпълнява, ако е върнат резултат **true** от условието?
    - а) блока описан след **else**
    - б) винаги се изпълнява **if**
    - в) нито един от двата
    - г) и двата блока
    - д) изпълняват се командите описани в блока **label**
  5. В структурата **if (условие) else**, кой блок с команди се изпълнява, ако е върнат резултат **false** от условието?
    - а) блока описан след **else**
    - б) винаги се изпълнява **if**
    - в) нито един от двата
    - г) и двата блока
    - д) изпълняват се командите описани след командалата **label**
- Верни отговори: 1 (а, в); 2 (б, в); 3 (б); 4 (б); 5 (а)*



## Цикли в управляващите програми

**КАКВО НАЛАГА  
НЕОБХОДИМОСТ  
ТА ОТ  
ИЗПОЛЗВАНЕТО  
НА ЦИКЛИ**

**КАК ДА  
СЪЗДАВАМЕ  
И  
ИЗПОЛЗВАМЕ  
НА ЦИКЛИ**

Много често при писането на програми се налага определен код от програмата да се изпълнява многократно. Итерациите или повторенията могат да бъдат точен брой пъти, определен предварително, но също така те могат да бъдат неопределен брой пъти или зависещи от външен сигнал за контролера. Това налага използването на циклична структура в програмата. Така общото количество от командни редове намалява рязко, а освен това и четимостта на програмата става по-ясна и разбираема. Използването на цикли в дадена програма се явява изключително мощен и полезен инструмент в програмирането.

Циклите биват няколко вида:

### Цикли с предварително зададени параметри

Това са най-често използваните циклични структури. Броят на повторенията обикновено се задава в самото организиране на цикъла. Този вид цикли се създава с оператора **for**.

На фиг. 15 е показана структура на цикъл с оператора **for**.

```
44  for(i = 0; i < 10; i=i+1){  
45      operator 1  
46      operator 2  
47      ...  
48      ...  
49      operator n  
50 }
```

Фиг. 15: Синтаксис на оператора for

В структурата на цикъла с оператора **for**, в големи скоби {} се поставя кодът, който ще се повтаря (ред 45 до 49 от фиг. 15) в зависимост от параметрите на цикъла. Входните данни или параметрите на цикъла се поставят в обикновени скоби (). За правилната работа на цикъла **for** са необходими три параметъра и индексна променлива, която участва в тях. Най-често индексната променлива е от тип **integer** (цяло число), но също така може да бъде и променлива от тип **float** (дробно число).

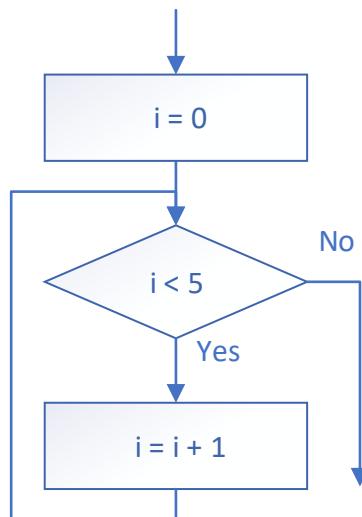
Тези три параметъра са разделени със символа точка и запетая (;) както е показано на ред 44 от фиг. 15.

Първият параметър е за начало на цикъла. В него се задава стойността, от която трябва да започне цикъльт. За целта на индексната променлива, която е предварително обявена, се задава стойността за начало. Стойността, която ѝ се задава може да бъде зададена както от константа (в случая числото 0), така и от друга променлива.

Вторият параметър е за край на цикъла. В него се описва някакъв логически израз за сравнение на индексната променлива (в случая i) с някаква друга променлива или константа. Този параметър задава края на повторенията в цикъла, които ще продължават до изпълнение на зададеното условие в логическия израз.



Третият параметър служи за определяне на стъпката, през която ще брои цикълът до достигане на края. Стъпката може да бъде зададена също с константа или друга променлива. Параметърът на стъпката представлява израз, в който участва индексната променлива  $i$ . Изразът се изчислява при всяка итерация на цикъла. На фиг. 16 е показана блокова схема на алгоритъм за реализиране на цикъл.



Фиг. 16: Блокова схема на алгоритъм за реализиране на цикъл

Как работи този цикъл?

При навлизане в него и изпълнение на оператора **for**, променливата  $i$  приема стойност **0**. Проверява се условието на параметъра за край на цикъла и ако то е изпълнено се изпълнява блокът на цикъла, в който са поставени съответните командни редове (ред 45 до 49 от фиг. 15.). След изпълнение на последния ред (ред 49) от блока на цикъла, управлението се връща в началото на цикъла и се изпълнява (изчислява се) изразът, който е описан в третия параметър (в случая  $i = i + 1$ ). Изразът ( $i = i + 1$ ) се чете така: към старото съдържание на променливата  $i$  (което при първата итерация е **0**), се прибавя стойността 1 и новата стойност се записва в същата променлива  $i$ . След изчислението, променливата  $i$  получава стойност **1**. Извършва се отново проверка дали резултатът от условието в израза на втория параметър ( $i < 10$ ) е изпълнено. Условието ( $i < 10$ ) се чете така, докато  $i$  е по-малко от **10**, условието е изпълнено. Резултатът, който ще бъде върнат при тази итерация от логическия израз е **true**. В случая  $i = 1$  и условието е изпълнено, тъй като **1** е по-малко от **10** и тялото на цикъла се изпълнява отново. След всяка итерация променливата  $i$  се увеличава с **1** и ще бъде съответно **0, 1, 2, 3, 4, 5, 6, 7, 8** и **9**. Повторенията продължават докато резултатът, който се връща от логическия израз не бъде **false**. След изпълнение на последната итерация, която ще бъде с  $i = 9$ , стойността на  $i$  ще се увеличи с **1** и ще бъде  $i = 10$ . Тогава резултатът от проверката на условието ще бъде **false**. При резултат **false**, се излиза от цикъла (тялото на цикъла не се изпълнява) и управлението се предава на програмните редове, които са след

цикъла (в случая ред 51 от фиг. 15).

### Цикли с предусловие

Условието се задава в началото на цикъла. Особеното при този вид цикли е, че цикълът може да не бъде изпълнен нито веднъж. Това е възможно поради факта, че първо се извършва проверка на зададено условие и ако то е изпълнено (ако връща **true**), тогава се влиза и се изпълнява тялото на цикъла (блока с команди заграден с големи скоби **{}**). Друга особеност е, че изразът, който променя стойността на индексната променлива с указаната стъпка, се поставя вътре в тялото на цикъла. Този вид цикли се създава с оператора **while**. Началната стойност на индексната променлива на цикъла трябва да се зададе извън самия цикъл (преди оператора **while**). Този вид цикли са много удобни за проверка на външен сигнал, подаван към микроконтролера.

На фиг. 17 е показана структура на цикъл с оператора **while**.

```

53 | i = 0;
54 | while(i < 10) {
55 |   operator 1
56 |   operator 2
57 |   ...
58 |   ...
59 |   operator n
60 |   i = i + 1
61 |

```

Фиг. 17: Синтаксис на оператора **while**

В структурата на цикъла с оператора **while** в големи скоби **{}** се поставя кодът (ред 55 до 60 от фиг. 17), който ще се изпълнява при всяка итерация в зависимост от условието на цикъла. Условието на цикъла се поставя в обикновени скоби **()** непосредствено след оператора **while**. За правилната работа на цикъла **while** е необходимо стойността на индексната променлива, която участва в условието на цикъла, да се зададе преди влизане в цикъла (ред 53 от фиг. 17). Както при предния цикъл с **for**, така и при цикъла с **while**, най-често индексната променлива е от тип **integer** (цяло число), но също така може да бъде и променлива от тип **float** (дробно число).

Как работи цикълът с оператора **while**?

Индексната променлива получава своята начална стойност преди навлизане в цикъла. В случая тя приема стойност **0** (ред 53 от фиг. 17). Следващата стъпка е да се провери условието на цикъла заградено в малки скоби **()**. Ако резултатът от условието е **true**, то тогава се влиза в цикъла. В противен случай (ако върнатият резултат е **false**) тялото на цикъла се прескача и изпълнението на програмата продължава с командите описани след цикъла (от ред 62 на фиг. 17).

При върнат резултат **true**, се влиза в цикъла и се изпълняват командите описани в него (ред 55 до 60 от фиг. 17). Важно условие е в тялото на цикъла да се намира команда за промяна (в случая увеличение с 1) на индексната променлива. В противен случай излизането от цикъла няма да бъде възможно, което ще доведе до

безкраен цикъл и блокиране на програмата. След изпълнение на последния ред от тялото на цикъла, управлението се връща в началото за следваща итерация. Отново се проверява условието и в зависимост от резултата, действията се повтарят или се излиза от цикъла.

### Цикли със следусловие

Синтаксисът на такъв цикъл започва с оператора ***do*** следван от тялото на цикъла заградено в големи скоби ***{}***. Условието се задава в края на цикъла в обикновени скоби ***()***, след оператора ***while***. Особеното при този вид цикли е, че цикълът винаги ще бъде изпълнен поне веднъж. Това се случва поради факта, че първо се изпълнява тялото на цикъла и тогава се проверява условието след него.

На фиг. 18 е показана структура на цикъл с оператора ***do .. while***.

```

64 i = 0;
65 do {
66     operator 1
67     operator 2
68     ...
69     ...
70     operator n
71     i = i + 1
72 } while(i < 10);

```

Фиг. 18: Синтаксис на оператора ***do .. while***

Как работи цикълът с оператора ***do .. while***?

Както и в предните типове цикли, тук индексната променлива също получава своята начална стойност преди навлизане в цикъла. В случая тя приема стойност ***0*** (ред 64 от фиг. 18). Следващата стъпка е да се изпълнят операторите, намиращи се в тялото на цикъла (ред 66 до ред 71 на фиг. 18). Някъде в цикъла трябва да съществува ред с команда за промяна стойността на индексната променлива. В случая това се изпълнява на ред 71 от фиг. 18. В противен случай индексната променлива няма да промени своята стойност, условието няма да върне ***false*** и ще се получи безкраен цикъл, т.е. програмата няма да може да излезе от него. Накрая се проверява условието на цикъла заградено в малки скоби ***()*** след оператора ***while*** (ред 72 от фиг. 18). Условието се чете така: докато резултатът от условието е ***true***, тогава повтори тялото на цикъла отново. Ако резултатът от условието е ***false***, тогава цикълът се напуска и управлението се предава на операторите след него. В случая това е ред 73 от фиг. 18.

### ОЦЕНКА НА ЗНАНИЯТА

1. Колко итерации ще направи следният цикъл: `for(i=0; i<=5; i++)?`

- a) 6
- б) 5
- в) 0
- г) 3

2. Докога ще се изпълнява цикълът със следусловие?

- а) изпълнява се веднъж и излиза
- б) докогато променливата достигне максимална стойност
- в) докато резултатът от условието е истина
- г) докогато има прекъсване
- д) докогато операторът натисне случаен бутон

3. Докога ще се изпълнява цикълът с предусловие?

- а) изпълнява се веднъж и излиза
- б) докогато стойността на променливата бъде равна на 0
- в) докато резултатът от условието е истина
- г) докогато има прекъсване
- д) докогато операторът натисне случаен бутон

4. Кое от твърденията е вярно за цикъл с предусловие?

- а) изпълнява се най-малко веднъж
- б) винаги се изпълнява
- в) може и нито веднъж да не се изпълни
- г) излиза се от него когато се изпълни условието
- д) изпълнява се най-много веднъж
- е) влиза се в него циклично

5. Кое от твърденията е вярно за цикъл със следусловие?

- а) изпълнява се само ако условието е истина
- а) изпълнява се най-малко веднъж
- в) може и нито веднъж да не се изпълни
- г) излиза се от него когато се изпълни условието
- д) изпълнява се най-много веднъж
- е) влиза се в него циклично

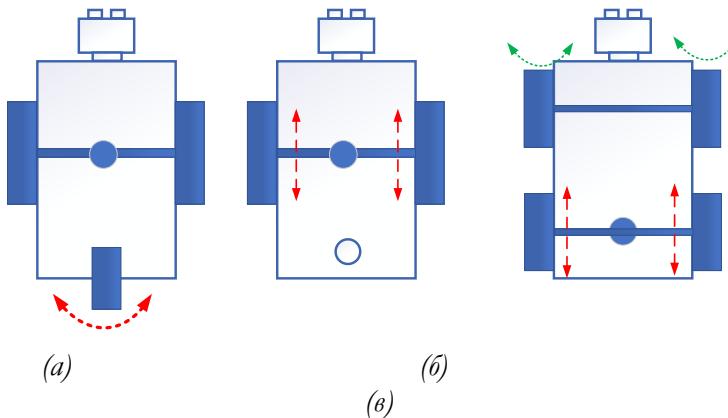
*Верни отговори: 1 (а); 2 (б); 3 (в); 4 (в); 5 (б)*



## Движения и позициониране на робота

### ХАРАКТЕРИСТИКИ НА МОБИЛНИТЕ РОБОТИ

Роботите не могат да виждат или да чувстват като хората, но ако са оборудвани със сензори, те могат да получават и информация за околната среда. Ако програмата, интегрирана в робота, може да интерпретира тази информацията, то тогава роботът може да реагира на определени въздействия.



*Колесен робот: единично задвижващо колело (а), диференциално задвижване (б) и акерман управление (в)*

Дизайнът на платформата на (а) има единично задвижващо колело, което е и завиващо. Необходими са два двигателя, един за задвижване и един за завиване. Предимството на тази конструкция е, че изпълнението на двете функции движение и завиване са напълно разделени чрез използването на два двигателя. Следователно, управляващият софтуер за задвижване и завиване няма да е сложен. Недостатък на тази конструкция е, че роботът не може да се върти на място, тъй като задвижващото колело не е разположено в центъра на платформата.

Дизайнът на платформата на (б) се нарича „диференциално задвижване“ и е една от най-често използваните конструкции за мобилни роботи. Комбинацията от две задвижващи колела позволява роботът да се движи напред, да завива и да се върти на място. Връзката между командите за управление, например крива на даден радиус и съответните скорости на колелата, трябва да се осъществи посредством софтуер. Друго предимство на тази конструкция е, че двигателите и колелата са с фиксирани позиции. Това значително опростява дизайна на механиката на робота.

Платформата (в) е платформа с така нареченото „акерман управление“, която е стандартно задвижвана и е със завиваща система на кола със задно задвижване. Има един двигател, който задвижва и двете задни колела с помощта на диференциална кутия и един двигател за управление на завиването на двете предни колела.

Необходимо е да се отбележи, че и трите типа конструкции на мобилните платформи изискват два двигателя, които са достатъчни за задвижване и завиване.

Един много голям недостатък на колесните роботи е, че те изискват

сравнително гладки повърхности, по които да се движат. Верижните роботи, представени на фигурата по-долу (б) са по-гъвкави и могат да се движат по неравен терен. Имат голяма проходимост като движение нагоре и надолу или преодоляване на препятствия и са лесни за управление. Недостатък е, че нямат точността на колесен робот, но с необходимите системи и сензори са надеждни. Верижните платформи се нуждаят от две двигателя, по един за всяка верига.



a

б

в

*Колесен (а), верижен (б) и крачещ (в) робот*

Крачещите роботи (в), подобно на верижните роботи могат да се движат по неравен терен и да преодоляват препятствия. Главното правило е: колкото повече крака, толкова по лесен баланс. Например шест-кракия робот (в) може да се управлява по такъв начин, че три от краката винаги да са на земята, докато другите три са във въздуха. Крачещите роботи обикновено изискват по два или повече двигатели на крак, така че на шест-краковия робот са необходими 12 двигателя. Използването на повече двигатели позволява по-голяма свобода на движенията, но за сметка на тежестта на платформата и цената на частите.

## УПРАВЛЕНИЕ НА ДВИГАТЕЛИТЕ

### Управление без обратна връзка

В задачите за управление обикновено има два обекта: управляващ (мастер или главен) и управляван (подчинен). В най-прости случаи, команда идва от главното устройство, а подчиненото я изпълнява без да съобщава нищо за резултата на управляващото устройство. В това се състои същността на управлението без обратна връзка.



*Управление без обратна връзка*

От гледна точка на мобилния робот, управляващото устройство е неговият контролер с изпълняваната програма, обектът на управление са неговите колела и тяло (шаси). Контролерът подава

командите за управление на двигателите в определена последователност, синхронизирани по време с вътрешния му часовник, наречен таймер.

Първият клас задачи, от които започва програмирането, е управление движениета на робота. Ще ги разгледаме последователно.

- *Движение напред и назад за определено време*

За движение напред се използват команди за управление на двигателя. Тези команди просто включват двигателите за определено време. Например:

```
task main() {
    motor[motorB] = 100; // motors forward
    motor[motorC] = 100; // with maximum power
}
```

Тук командалата `motor[]` е масив от три елемента, всеки от които определя текущото състояние на въртене на двигатели А, В и С. Действието “full forward” се дава с число 100, действието “full back” с -100, а стопът е 0. Действието на тази команда може да се счита за незабавно. След нейното изпълнение двигателят се включва и продължава да работи, докато не бъде спрян от друга подобна команда.

Ако веднага след тях се изпълни команда за изключване на двигателите, роботът няма да се движи, а ще стои неподвижен. Ето и един малък пример:

```
task main() {
    motor[motorB] = 100; // motors forward
    motor[motorC] = 100; // with maximum power
    motor[motorB] = 0; // stops the engines
    motor[motorC] = 0;
}
```

По този начин, за осъществяването на движението е необходимо известно закъснение преди изключване на двигателите. Функциите за изчакване не произвеждат конкретни действия, но позволяват на двигателите да изпълняват своята част от работата.

```
task main() {
    motor[motorB] = 100; // motors forward
    motor[motorC] = 100; // with maximum power
    wait1Msec(1000); // is delay
    motor[motorB] = 0; // stops the engines
    motor[motorC] = 0;
}
```

Движението напред или назад се определя от посоката на въртене на двигателите. За да се зададе посоката на движение при EV3 роботи се задава положителна или отрицателна стойност на скоростта за движение. Например:

```
task main() {
    motor[motorB] = 100; // motors forward
    motor[motorC] = 100; // with maximum power
    wait1Msec(1000); // 1s delay
    motor[motorB] = -100; // motors backward
    motor[motorC] = -100; // with maximum power
    wait1Msec(1000); // 1s delay
    motor[motorB] = 0; // stops the engines
    motor[motorC] = 0;
}
```

В момента на промяна на посоката при висока скорост е възможно приплъзване. Затова се препоръчва плавно спиране. За да направите това, преди да дадете команда “назад”, двигателите се изключват от напрежение и роботът за известно време се движи по инерция. Например:

```
task main() {
    motor[motorB] = 100; // motors forward
    motor[motorC] = 100; // with maximum power
    wait1Msec(800); // 800ms delay
    // Includes floating mode for motor control
    bFloatDuringInactiveMotorPWM = true;
    motor[motorB] = 0; // stops the engines
    motor[motorC] = 0;
    wait1Msec(200); // 200ms delay
    motor[motorB] = -100; // motors backward
    motor[motorC] = -100; // with maximum power
    wait1Msec(1000); // 1s delay
    // Starts brake mode for motor control
    bFloatDuringInactiveMotorPWM = false;
    motor[motorB] = 0; // stops the engines
    motor[motorC] = 0;
}
```

#### • Завиване

За да се извърши завой на място, е достатъчно да се завъртят двигателите в различни посоки. След това роботът ще се върти приблизително около центъра на оста на задвижващите колела, изместен към центъра на тежестта. За по-точен завой, трябва да изберете времето в стотни от секундата. Въпреки това, когато се промени зарядът на батерията, ще трябва да се въведат нови параметри на въртене:

```
task main() {
    motor[motorB] = 100; //motors are moving in different directions
    motor[motorC] = -100; // with maximum power
    wait1Msec(300); // 300ms delay
    motor[motorB] = 0; // stops the engines
    motor[motorC] = 0;
}
```

Има и друг вид завои. Ако един от двигателите е спрян или му е зададена по-ниска скорост на движение, а другият е включен, въртенето ще се случи около по-бавния мотор. Тогава въртенето ще бъде по-плавно:

```
task main() {
    motor[motorB] = 100; // the motor runs at maximum power
    motor[motorC] = 0; // the motor is stopped
    wait1Msec(1000); // 1s delay
    motor[motorB] = 0; // stops the rotation of motor B
}
```

#### • Движение по квадрат

Използвайки получените знания за управление на двигателя, е възможно да се програмира движение по квадрат или друг многоъгълник с помощта на цикъл или безусловен преход.

*Подкрепата на Европейската комисия за изготвянето на настоящата публикация не представлява одобрение на съдържанието, което отразява гледните точки само на авторите и не може да се търси отговорност от Комисията за всяка употреба, която може да бъде използвана за информацията, съдържаща се в нея.*



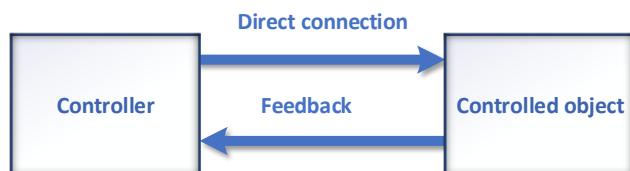
```
task main() {
    motor[motorB] = 100; // forward move
    motor[motorC] = 100;
    wait1Msec(1000);
    motor[motorB] = 100; // turn 90 degree
    motor[motorC] = 0;
    wait1Msec(500);
    motor[motorB] = 0;
}
```

След като определите продължителността на завъртанията и броя на повторенията, ще научим робота да върви по периметъра на квадрата или фигурата за обхождане. За получаване на прецизно завиване, трябва да се намали мощността на двигателите около два пъти. Закъсненията ще трябва да се настройт в процеса на тестване.

```
task main() {
    for(int i=0;i<4;i++){
        motor[motorB] = 50; // forward move
        motor[motorC] = 50;
        wait1Msec(1000);
        motor[motorB] = 50; // turn 90 degree
        motor[motorC] = 0;
        wait1Msec(400);
    }
    motor[motorB] = 0;
}
```

### Управление с обратна връзка

Появата на обратна връзка в системата означава, че управляващото устройство започва да получава информация за обекта на управление. Обратната връзка се осъществява с помощта на сензори, прикрепени към тялото на робота. Данните постъпват в контролера, който представлява управляващо устройство.



Управление с обратна връзка

Например, за да не зависи движението от заряда на батерията, е възможно да се използва сензор за следене за оборотите на двигателя (енкодер), който позволява да се правят измервания с точност от 1 градус. В следващия пример се използва обстоятелството, че когато количката се завърти на 90 градуса, лявото колело се върти на 250 градуса около оста си.

```
task main() {
    nMotorEncoder[motorB]=0; // initialization of the encoder
    motor[motorB] = 50; // forward move
    motor[motorC] = 50;
    // Waiting for reading of encoder
    while(nMotorEncoder[motorB]<250){
        motor[motorB] = 0;
        motor[motorC] = 0;
    }
}
```



## Сензори в роботите

**ЗАЩО  
РОБОТИТЕ  
ИМАТ НУЖДА  
ОТ СЕНЗОРИ?**

**СЕРВОМОТОР.  
СЕНЗОР ЗА  
ВЪРТЕНЕ НА  
ДВИГАТЕЛЯ  
(ЕНКОДЕР)**

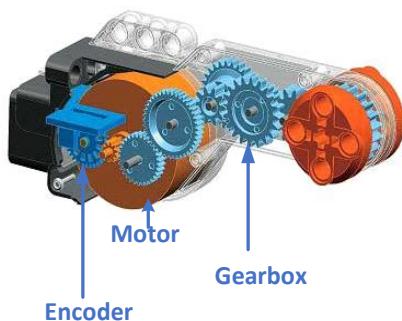
Роботите не могат да виждат или да чувстват като хората, но ако са оборудвани със сензори, те могат да получават и информация за околната среда. Ако програмата, интегрирана в робота, може да интерпретира тази информацията, то тогава роботът може да реагира на определени въздействия.

Сервомоторът на Lego Mindstorms EV3 е комбинация от електрически двигател, редуктор и ротационен сензор, комбинирани в един корпус със специфична форма.



*Сервомотор на Lego Mindstorms EV3*

Малогабаритните постояннотокови двигатели обикновено се върят твърде бързо и нямат мощност на вала, така че няма смисъл да се свързват директно към колела или други механизми. За да се намали скоростта на въртене и в същото време да се увеличи въртящият момент, обикновено се използва някаква предавка. Серводвигателят съдържа осем предавки с общо предавателно отношение 1: 48.



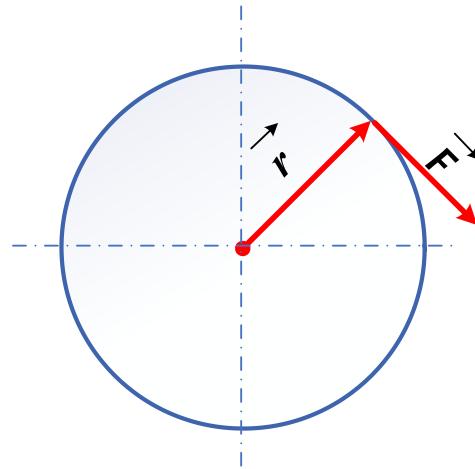
*Вътрешно устройство на сервомотор на EV3*

Във всеки двигател е вграден сензор за въртене (енкодер). Той позволява да се контролират по-точно оборотите на двигателите. Сензорът за въртене измерва ъгъла на завъртане на двигателя в градуси с точност от  $\pm 1^\circ$ . Пълният оборот на двигателя е  $360^\circ$ . Енкодерът позволява да се регулират скоростите за движение на двигателите.

Една от най-важните характеристики на двигателя е въртящият момент (момент на сила или само момент). Моментът на силата е векторна физическа величина, равна на произведението на радиус-вектора, спуснат от оста на въртене до точката на

*Подкрепата на Европейската комисия за изготвянето на настоящата публикация не представлява одобрение на съдържанието, което отразява гледните точки само на авторите и не може да се търси отговорност от Комисията за всяка употреба, която може да бъде използвана за информацията, съдържаща се в нея.*

прилагане на силата, с вектора на тази сила.

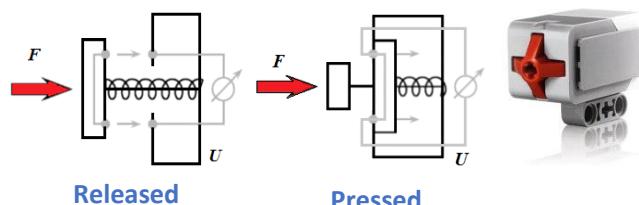


*Момент на силата*

#### СЕНЗОР ЗА ДОПИР

Докосването и усещането на натиск от робота може да се осъществи с помощта на сензор за допир (Touch Sensor). Това е бутон, който има две възможни състояния - „натиснат“ (*pressed*) и „освободен“ (*released*). Програмно, сензорът разпознава и друго състояние на „кликнат“ (*Bumpted*).

Сензорите за допир обикновено се използват в роботите за откриване на препятствия по пътя за движение.

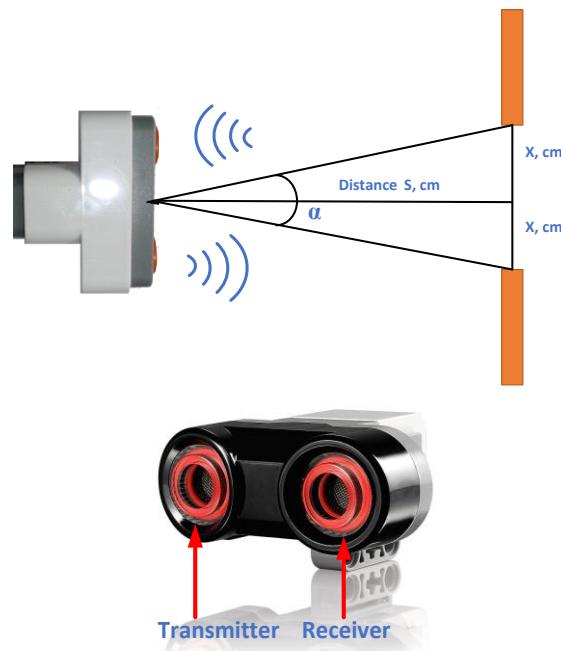


*Схема и външен вид на сензор за допир в Lego Mindstorms EV3*

#### СЕНЗОР ЗА РАЗСТОЯНИЕ

Ултразвуковият сензор дава възможност на робота да вижда и разпознава обекти, да избягва препятствия, да измерва разстоянието и да открива движение. Сензорът измерва разстоянието до обекта в сантиметри и инчове. Диапазонът на измерване е от 0 до 2,5m с точност от  $\pm 3$  см.

Сензорът за разстоянието се състои от предавател и приемник. Предавателят излъчва вълна в ултразвуковия диапазон, вълната се отразява от повърхността на обекта и се приема от приемника (виж фигураната по-долу).



Функционална схема на сензора за разстояние

За да измине звуковата вълна разстоянието до обекта ( $S$ ) и обратно е необходимо време ( $\Delta t$ ). Затова програмата няма да получи веднага данните от сензора, а след този момент. Ултразвуковият сензор от комплекта Lego Mindstorms EV3, позволява да се определи разстоянието  $S$  до 2,5 метра. Това означава, че знаейки скоростта на звука във въздух  $V_{\text{звук}}$  ( $V_{\text{звук}} = 330 \text{ m/s}$ ), можем да изчислим, че максималното време на изчакване за обратната вълна  $\Delta t$  е:

$$\Delta t = \frac{V_{\text{звук}}}{2 \cdot s} = \frac{330 \text{ m/s}}{2 \cdot 2,5 \text{ m}} = \frac{1}{66} \text{ s} \approx 0,15 \text{ s}$$

Големите обекти с твърда, добре отразяваша звукова повърхност дават най-надеждни резултати. Най-лошите резултати се получават при обличване на малки или тънки предмети с извита повърхност (например топка). Меките предмети могат да абсорбират ултразвуковите вълни и да не бъдат открити от този сензор. Два или повече ултразвукови сензора, работещи в едно и също помещение, могат да си пречат и да намалят точността на резултатите от измерването.

## СЕНЗОР ЗА СВЕТЛИНА

Осветеността е физическа величина, числено равна на количеството светлинен поток попаднал на единица площ. В система SI единицата за осветеност се нарича люкс ( $\text{lx}$ ). Осветеността е правопропорционална на интензитета на светлината на източника, зависи от разстоянието на светлинния източник до осветения обект, както и от тъгъла, под който светлината пада върху повърхността.

Съществуват голям брой различни типове светлинни сензори като фоторезистори, фотодиоди и фототранзистори. Сензорът за

*Подкрепата на Европейската комисия за изготвянето на настоящата публикация не представлява одобрение на съдържанието, което отразява гледните точки само на авторите и не може да се твърди отговорност от Комисията за всяка употреба, която може да бъде използвана за информацията, съдържаща се в нея.*

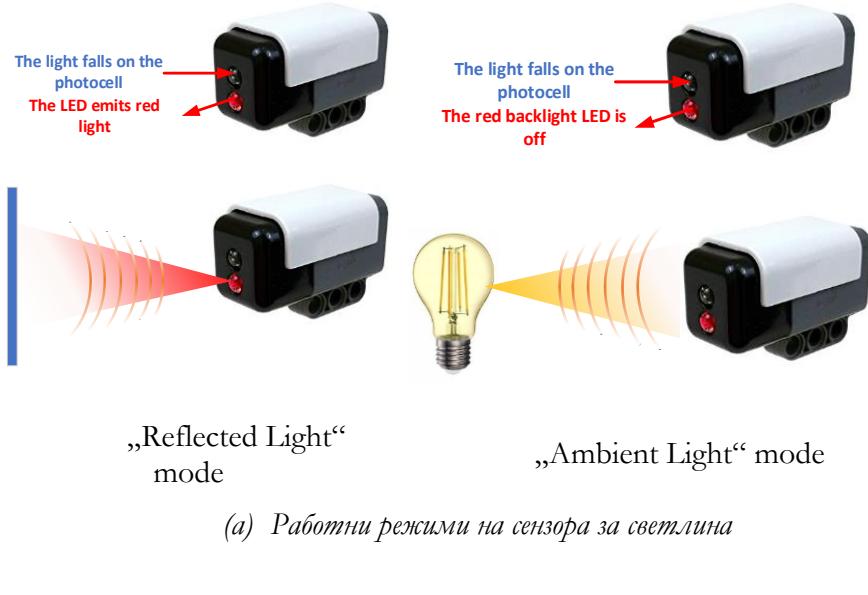
светлина (LightSensor) съдържа инфрачервен светодиод, който може да се включва и изключва програмно, както и фотоклетка, която измерва яркостта на светлината, падаща върху нея (виж фигурата по-долу). В режим на измерване на околната светлина (Ambient Light) количеството светлина, което пада върху фоточувствителния елемент, се преобразува в цифрова стойност, която вече се използва в програмата. Например, със сензор, работещ в този режим, може да се конструира робот, който търси най-осветеното място в стаята.

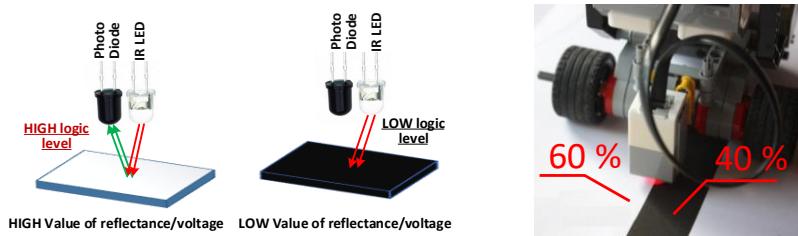


*Сензор за светлина*

В режим на измерване на отразения цвят (Reflected Light), към светочувствителния елемент е добавен източник на светлина (LED). Излъчената светлина се отразява от някаква повърхност и се връща обратно във фоточувствителния елемент.

Колкото по-светла е отразяващата повърхност, толкова повече светлина идва във фоточувствителния елемент. И обратно, колкото по-тъмна е повърхността, толкова по-малко светлина идва във фоточувствителния елемент. Това количество светлина се преобразува в цифрова стойност и се предава на програмата. Ако сензорът е разположен над тъмна повърхност, в програмата се получават малки стойности. Колкото по-ярка е повърхността, толкова повече светлина се отразява към сензора (виж фигура (b) по-долу). За правилна работа на сензора е необходимо той да е разположен под прав ъгъл на разстояние 1 см над повърхността.





(b) Показания на светлинния сензор (в режим на отразена светлина)  
на светла и на тъмна повърхност

Фотоклетката в светлинния сензор е по-чувствителна към инфрачервеното лъчение, отколкото към обичайния спектър на видимата светлина.

Един от начините за управление движението на роботи е те да се движат по маркировка. За целта се използва светлинен сензор, който се монтира вертикално към пода, върху който се намира маркировката. За правилната работа на алгоритъма за движение по линия е необходимо да се знае праговата стойност на реакцията на сензора. Праговата стойност е число, което разделя два диапазона от стойности, например "светло" и "тъмно". Използвайки праговата стойност, роботът определя кои стойности на показанията на светлинния сензор са за „светло“ и кои са за „тъмно“. Това позволява да се разделят всички възможни показания на сензора (от 0 до 100) на два диапазона. В зависимост от това кой от диапазоните следва, роботът завива наляво или надясно, за да не загуби линията. Праговата стойност за различните случаи ще се променя. Тя може да се окаже твърде висока или твърде ниска в зависимост от нивото на осветеност в помещението, както и от отражателната способност на повърхността на работното поле и черната линия (маркировката). Праговата стойност се определя като средно аритметично между показанията на сензора в тъмните и светлите зони.

#### СЕНЗОР ЗА ЦВЯТ

Сензорът за цвят (Color Sensor) от Lego Mindstorms EV3 е усъвършенстван светлинен сензор и може да разпознава 6 различни цвята. В допълнение към основната си задача - да различава цветовете, този сензор напълно дублира функциите на светлинен сензор. Сензорът разпознава цветовете, като измерва последователно отразената светлина от червени, зелени и сини светодиоди. Резултатите от тези измервания се обработват и сензорът определя най-близката стойност на цветовете под формата на код. За да се определи правилно цветът, е необходимо сензорът да бъде под прав ъгъл на разстояние 1 см над повърхността.



*Сензор за цвет на Lego Mindstorms EV3*

#### СЕНЗОР ЗА ЗВУК

Звуковият сензор (*Sound Sensor*) представлява микрофон. Той измерва нивото на звуковия сигнал (звуци от 20-20000 Hz, възприемани от човешкото ухо), а също така инфразвук (под 20 Hz) и ултразвук (над 20000 Hz)]. Силата на звука или нивото на звуковото налягане се измерва в единици, наречени децибели (dB). Децибелите показват колко е по-висок или по-нисък звукът в сравнение с друг звук. В този случай 0 dB е най-тихият звук на границата на слуха на обикновен човек. Максималното звуково налягане, което сензорът може да измери е около 90 dB, което съответства на силата на звука на косачка. Показанията на звуковия сензор се изразяват като процент от максималната сила, която може да се измери.



*Звуков сензор на Lego Mindstorms EV3*

Звуковият сензор на Lego Mindstorms EV3 дава следните показания: 4-5% съответстват на нивото на шума в тихата всекидневна стая; 5-10% - обикновена реч, чута на средно разстояние; 10-30% - нормален разговор в близост до сензора или музика, възпроизведена на нормално ниво на звука; 30-100% - силни писъци или музика. Данните са валидни когато сензорът е разположен на разстояние 1 m от източника на звука. Звуковата чувствителност на човешкото ухо е силно зависима от честотата на звука. Пикът на чувствителността към звука е 3-5 kHz, а стойностите под 20 Hz и над 20 kHz вече са извън физическите възможности. Звуковият сензор може да бъде превключен в режим dBA, при който стойностите на измерването най-точно съответстват на диаграмата за чувствителност на човешкото ухо и съответната стойност на звуковото налягане.

## СЕНЗОР ЗА УСКОРЕНИЕ (АКСЕЛЕРОМЕТЪР)

Акселерометрите са миниатюрни сензори за движение, които се използват в различни устройства, включително смартфоните и таблетите. Те измерват ускорението – величина, която показва изменението на скоростта на движещ се обект за единица време. Единицата за ускорение е метър в секунда на квадрат ( $m/s^2$ ), като земното ускорение се означава с буквата  $g$  (от „гравитация“) и е равно на  $9,8 m/s^2$ .

Акселерометрите представляват електромеханични устройства, които измерват статичните сили (като гравитацията) или динамичните сили (към които спадат вибрациите и движението). Тези устройства могат да измерват ускорението по една, две или три оси, като акселерометрите в съвременните смартфони и таблети обикновено работят с три оси.

Сензорът за ускорение (*Accelerometer Sensor*) позволява на робота да определи резултатното ускорение при промяна на скоростта във всяка от трите посоки, както и да определи своето "отклонение" от хоризонталата.

Действието на сензора се основава на пиезоелектричния ефект – появата на електрически заряд на повърхността на пиезоелектрик под действието на механични напрежения. Когато работът променя скоростта (ускорява или забавя), неговата инерция позволява на пиезоелектричния елемент да се разтяга или свива. Знаейки масата  $m$ , дължината  $l$ , коефициента на пропорционалност  $q$  и измерването на напрежението  $U$ , ускорението може да се изчисли по формулата:

$$U \cdot q = m \cdot l \cdot a \Rightarrow a = \frac{U \cdot q}{m \cdot l}$$



*Принцип на действие и външен вид на сензора за ускорение*

Сензорът за ускорение позволява измерването на ускорението в диапазона от  $\pm 19,62 m/s^2$ , т.e. удвояване на ускорението за свободно падане  $g$  ( $g=9,81m/s^2$ ).

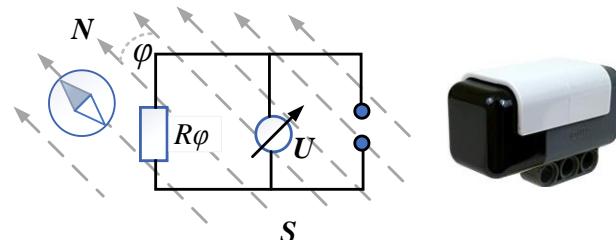
## СЕНЗОР- КОМПАС

Сензорът компас (*Magnetic Compass Sensor*) се използва да определи текущата ориентация на робота. Това е електронен аналог на обичайния компас, който дава информация за ориентацията в пространството в цифрова форма.

Работата на електронния компас (виж схемата по-долу) се основава на магниторезистивен ефект, т.e. промяна в

електрическото съпротивление на материал под действието на външно магнитно поле.

Магниторезистивното съпротивление  $R_\varphi$  променя своята стойност в зависимост от посоката на магнитните силови линии на полето, в резултат на което възниква потенциална разлика. Колкото по-голям е ъгълът на въртене на сензора  $\varphi$  спрямо силовите линии на магнитното поле на Земята, толкова по-голямо е съпротивлението. Точността на измерване на сензора е  $1^\circ$ , обхватът на измерените стойности е от 0 до  $359^\circ$ .



*Принцип на работа на сензор-компас*

#### ЖИРОСКОПИЧЕН СЕНЗОР

Lego Gyroscopic Sensor може да открие движение само около една ос на въртене. С помощта на жироскопичния сензор може да се определи скоростта на въртене в градуси и в секунди, което от своя страна може да се използва за определяне на времето на завъртане на робота или на негова част. В допълнение, жироскопичният сензор регистрира общия ъгъл на въртене в градуси. По този начин може да се контролира движението на робота. Също така, използвайки този сензор, може да се определи скоростта на въртене на робота в градуси в секунда.



*Външен вид на сензор жироскоп (NXT Gyro Sensor)*

Сензорът за определяне на скоростта на въртене е също едноосен жироскоп, свързан с кварцов резонатор. С него може да се изгради робот, който да реагира на скоростта на въртене. Сензорът NXT Gyro Sensor HiTechnic може да открива въртеливо движение на робота до скоростта на въртене от  $\pm 360^\circ/\text{s}$ . Измерената стойност е скоростта на въртене в градуси за секунда.

## ИНФРАЧЕРВЕН СЕНЗОР

Инфрачевреният сензор на Lego Mindstorms EV3 позволява да се откриват инфрачеврени сигнали, изпратени например от отдалечен инфрачеврен маяк (IR beacon), да изпраща собствен инфрачеврен сигнал и да определя отражението му от други обекти.

Инфрачеврено излъчване заема в електромагнитния спектър обхвата между червената граница на видимия спектър и микровълните (от 750 nm до 1 mm). Често се нарича топлинно излъчване. Този тип сигнал се използва в повечето телевизионни дистанционни управления. Подобно на видимата светлина, инфрачеврено лъчение се блокира от обекти по пътя си, така че между инфрачеврения излъчвател и инфрачеврения сензор трябва да има свободно пространство. Сънчевата светлина може да повлияе на инфрачеврените сигнали, но нормалното осветление в стаята няма да въздейства.

Сензорът за откриване на инфрачеврен източник с помощта на пет инфрачеврени сензора, насочени в различни посоки на всеки 60°, определя източника на IR сигналите и тяхната интензивност, както и генерира електромагнитно излъчване в инфрачеврения спектър.



*Инфрачеврен детектор и инфрачеврена топка*

## СЕНЗОР ЗА ОТКРИВАНЕ НА ОБЕКТИ

EOPD сензорът е електрооптичен детектор за близост. Той е подобен на стандартен светлинен сензор, с изключение на това, че използва импулсна светлина за да елиминира ефективно интерференцията на околната или фонова светлина при четене. Чрез използване на импулсна светлина, яркостта по време на импулса може да бъде относително висока. Сензорът отчита и определя разликата между измерването на светлината на детектора преди излъчването на импулса и измерването на светлината по време на импулса.

Сензорът EOPD има два режима на работа: чувствителност x1 и чувствителност x4. В режим на чувствителност x4, той може лесно да открие инфрачеврена топка на разстояние от 15 см. За близки бели обекти трябва да се използва режимът на чувствителност x1. Сензорът дава възможност да се конструира робот, който може да открива обекти, използвайки отразени светлинни сигнали. Вградената функция за корекция на околната светлина позволява да се използва както при ярка светлина, така и в затъмнени помещения.



*Сензор за отпирване на обекти*

## ОЦЕНКА НА ЗНАНИЯТА

1. Енкодерът е част от:
  - а) Серводвигател
  - б) Сензор за допир
  - в) Сензор за разстояние
  - г) Сензор за светлина
2. Сензорът за разстояние използва за определяне на разстоянието:
  - а) сигнали в диапазона 20-20000 Hz
  - б) сигнали в диапазона над 20 kHz
  - в) светлина в диапазона 380-740μm
  - г) светлина в диапазона над 740μm
  - д) светлина в диапазона под 380μm
3. Сензорът за измерване нивото на шума използва:
  - а) сигнали в диапазона 20-20000 Hz
  - б) сигнали в диапазона над 20 kHz
  - в) светлина в диапазона 380-740μm
  - г) светлина в диапазона над 740μm
  - д) светлина в диапазона под 380μm
4. Сензорът за определяне на цветове използва:
  - а) сигнали в диапазона 20-20000 Hz
  - б) сигнали в диапазона над 20 kHz
  - в) светлина в диапазона 380-740μm
  - г) светлина в диапазона над 740μm
  - д) светлина в диапазона под 380μm
5. Сензорът за следене на линия използва:
  - а) сигнали в диапазона 20-20000 Hz
  - б) сигнали в диапазона над 20 kHz
  - в) светлина в диапазона 380-740μm
  - г) светлина в диапазона над 740μm

а) светлина в диапазона под 380 $\mu$ m

6. Сензорът за откриване на обекти използва:

а) сигнали в диапазона 20-20000 Hz

б) сигнали в диапазона над 20 kHz

в) светлина в диапазона 380-740 $\mu$ m

г) светлина в диапазона над 740 $\mu$ m

д) светлина в диапазона под 380 $\mu$ m

Верни отговори: 1 (а); 2 (б); 3 (а); 4 (б); 5 (г); 6 (в)

## Анекс 1: Допълнителна информация

### Работи

#### КАКВО ПРЕДСТАВЛЯВА РОБОТЪТ

Работът е сложна автоматизирана система, която може да функционира както с участието и под управлението от човека, така и напълно самостоятелно. Той е устройство (стационарно или мобилно), което най-често имитира или наподобява действия, които обикновено се извършват от живи същества, човек или животно. Работът може да бъде и механична система, която се придвижва в пространството. За придвижването може да се използват обикновено колело, гъсенични вериги или лостове, наподобяващи крака на животни.

Думата „робот“ произлиза от чешката дума „*robot*“, означаваща работа. Тя е използвана за първи път от писателя Карел Чапек в пиесата му „R.U.R.“ (Росумски универсални роботи), публикувана през 1920 година.

Думата роботика (на английски: *robotics*), описваща научно-приложната област, изследваща роботите, е предложена от писателя Айзък Азимов.

Работът съчетава в себе си съвкупност от различни видове системи и модули събрани в едно цялостно устройство. Тези модули са създавани и са се използвали в различни други устройства още преди изобретяването и създаването на робота.

Освен основните му системи и модули, работът може да бъде надграждан и със специфични инструменти, чрез които да извърши необходимата работа (заваряващ апарат, система за лазерно рязане, механични пробивачи или режещи инструменти и др.).

Работът е колкото полезен, толкова и опасен за обкръжаващата го среда, както за хора и съоръжения така и за самите работещи с него. Високата скорост, с която се движки всяка една от осите и силата, която притежава, могат да нанесат както материални щети, а така също и да наредят околните и дори да доведат до инцидент с фатален изход. Затова безопасността е от съществено значение при използването на роботи. За да се предотвратят и избегнат евентуални наранявания, е необходимо да се спазват основни правила за безопасност при работа с робота.

#### ВИДОВЕ РОБОТИ

Думата робот се използва по отношение на голям набор от машини, които могат да извършват движение и различни операции. Те съществуват в множество форми: от хуманоид, който имитира човешката форма и начин на придвижване, до индустриски, чиито вид се определя от функцията, която изпълняват. Роботите могат да се класифицират като мобилни, индустриски и роботи, които са специализирани към задачата, която трябва да изпълняват.

На фигурата по-долу са показани различни видове роботи като домашни помощници, индустриски, военни, космически и хуманоидни.



*Роботи, използвани в различни области*

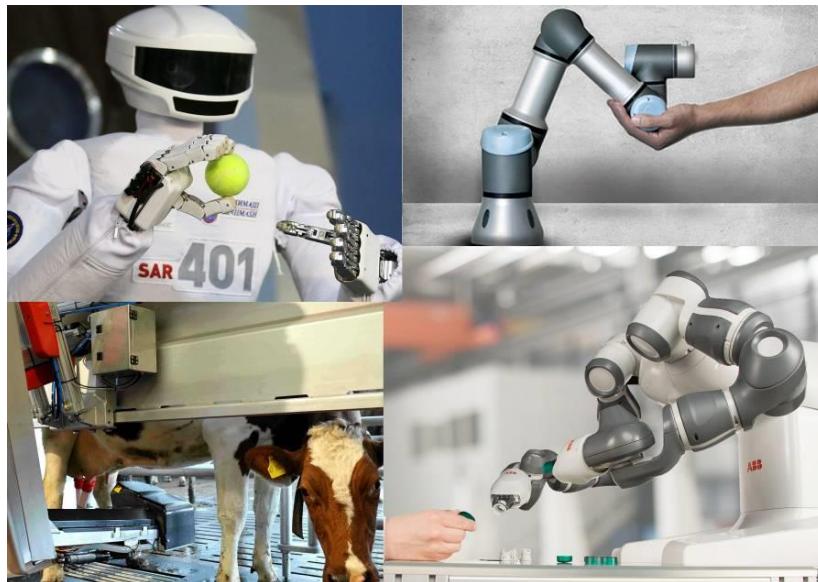
Работите управлявани от човек могат да бъдат използвани в различни отрасли като медицината, за военни цели, подводни апарати за оглеждане на морското дъно или състоянието на язовирни стени и съоръжения и др.

Работите, които работят самостоятелно без участието на човек по предварително зададена програма могат да бъдат: индустриски роботи, роботи за селското стопанство, за хранително-вкусовата индустрия, хуманоидни роботи за обслужване на хора с намалена подвижност и др.

Индустриските роботи са едни от най-разпространените, които се използват за автоматизирането на производствени процеси, като например: заваряване, боядисване, склобяване, хващане и преместване на обекти, изработка на печатни платки, пакетиране, етикетиране, инспекция, проверка на качеството и т.н. Всички тези дейности трябва да се изпълняват с висока надеждност, прецизност и ефективност. На фиг. 1.2 са показани различни видове индустриски роботи, при които не се допуска присъствие на хора в периметъра им на работа. Последните години интензивно се развиват като клас от индустриски роботи, така наречените *колаборативни роботи*. Те са разработени така, че да могат да работят в среда, в която има наличие на пресечни точки с хора или други машини. Роботът е програмиран така, че при появата на човек или възникването на препятствие в периметъра му на действие да намали скоростта си на движение, да изчака или да спре напълно своята работа, с цел избягване на злополука или авария. На фигура б по-долу са показани различни типове колаборативни роботи.



*a. Обикновени индустриални роботи*



*b. Колаборативни индустриални роботи*

#### ПРИЛОЖЕНИЕ НА РОБОТИТЕ

Роботите в днешно време могат да се срецне навсякъде, във всеки един отрасъл на икономиката и бита на хората. Роботите се използват за изпълнение на задачи, където работната среда е вредна и опасна за човека, като например при производството на интегрални схеми, производство на акумулятори, гасене на пожари, при ремонта на атомните реактори, при обезвреждане на мини и боеприпаси, в космоса и др.

Роботите намират приложение при дейности, при които се изисква прецизно движение и позициониране и необходимост от повторяемост на монотонни и еднотипни действия. Те постоянно се усъвършенстват с повече функции и възможности. Най-широко приложение роботите намират в автомобилостроенето. На фигурата по-долу са показани поточни линии от роботи произвеждащи роботи.



*Поточна линия от роботи произвеждащи роботи*

Работите могат да взаимодействат помежду си и да изпълняват разнородни задачи напълно самостоятелно без участието на хора. Полезността на поточна линия от роботи се състои в нейната гъвкавост и възможността от бърза промяна на вида на произвежданния продукт. При промяна и обновяване или пускане на нов модел автомобил, роботите от поточната линия могат да бъдат бързо и лесно програмирани и пренастроени в кратък срок, за да произвеждат новия продукт.

На фигурата по-долу са показани различни видове роботи, използвани от армията за разузнаване, обезвреждане, пренасяне на товари и др.

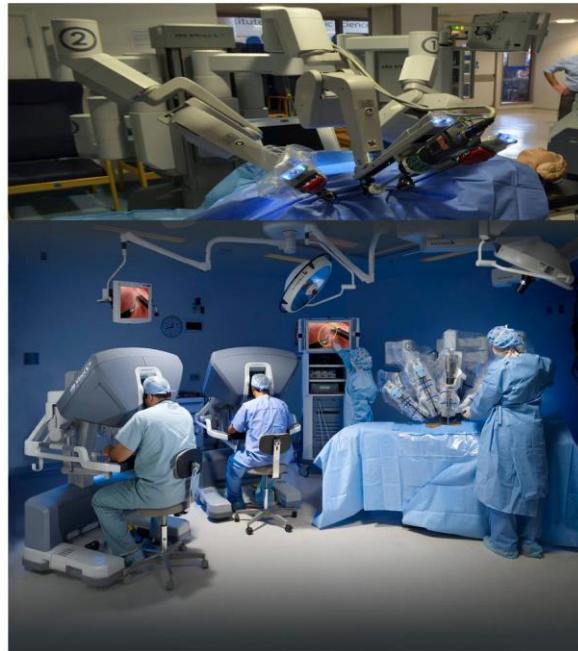


*Роботи, предназначени за военни цели*

Работите намират приложение и в бита и домакинството. Примери за това са машини и устройства които подпомагат или напълно замесят труда на хората при извършването на различни дейности в ежедневието. Това са роботизирани системи като прахосмукачки, машина за косене на трева и др.

Отдавна роботите са се наложили като незаменимо средство в науката при различни видове изследвания, както и при изучаването на океаните и космоса. Примери за това са увеличаващият се брой роботизирани системи, които са изпращани да изследват повърхността на Луната и на Марс.

Все по-трайно роботите намират приложение и в медицината. Използват се както при производството на различни лекарства, така и от лекари за операции на сложния човешки организъм, които изискват изключителна прецизност в действието на хирурга. Те не само дават възможност на хирурга да извършва необходимите действия, но могат и да го контролират и да не му позволяват да нарани околните тъкани по невнимание. На фигурата по-долу са показани роботи, с помощта на които се извършват сложни хирургически операции.



*Роботи, използвани в медицината*

Животът на хората е до такава степен обвързан с роботите, че е немислимо развитието на човешкото общество без тях.

## Алгоритъм в програмирането

Разработването на определен алгоритъм е първата стъпка, която трябва да се извърши, когато е необходимо да се напише програма.

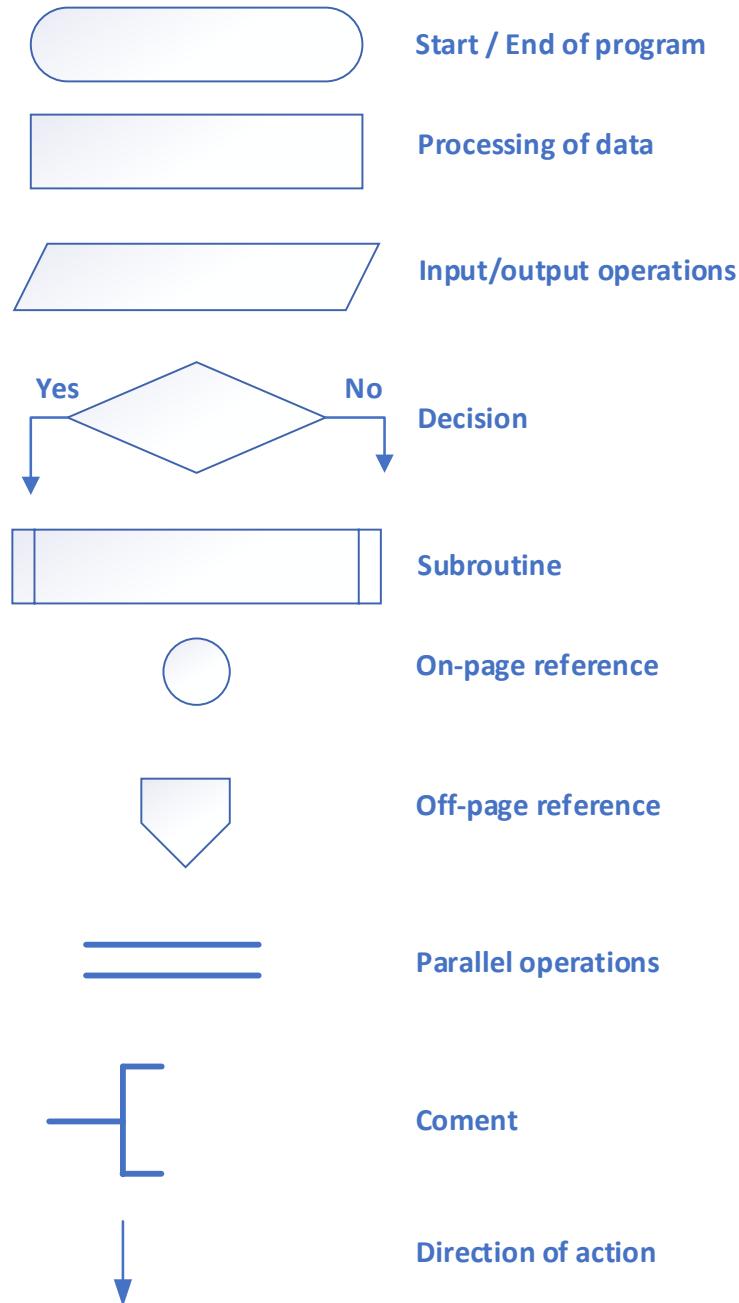
Много често, при създаването на дадена програма, програмистът сяда пред компютъра и без какъвто и да е алгоритъм започва да програмира, реализирайки някаква своя идея или задание. Това е напълно изгълната задача за програмисти с достатъчен опит, когато става въпрос за малки несложни програми. При реализиране на по-сложни програми, задължително условие е в началото да се отдели време за разработването на алгоритъм. Алгоритъмът трябва да реализира правилното функциониране на програмата. Много често при сложни програми не е достатъчно да се създаде само един главен алгоритъм, но и отделни алгоритми за различните функции, които ще се съдържат в общата програма.

При създаването на алгоритми се използват графични означения. Те се характеризират с това, че са лесни и удобни за използване. *Блоковата схема* е описание на алгоритъма, който се състои от начало, край, върхове (възли), разклоняващи и съединяващи ги клонове. Клоновете са отбелечани със стрелки, които показват посоката на движение, т.е. реда на изпълнение на операциите. На фигурата по-долу са показани основните графични символи, които се използват за изграждането на блокови схеми на алгоритми.

При съставяне на всеки алгоритъм трябва да се спазват следните правила:

- описание на стъпките в алгоритъма трябва да бъде ясно и точно и да не е ориентирано предварително към някакъв вид програмен език;
- действията при изпълнение на отделните стъпки не трябва да предизвикват двусмислие и да не подлежат на тълкуване;
- действията, описващи тялото на алгоритъма, трябва да имат крайно по обем описание.



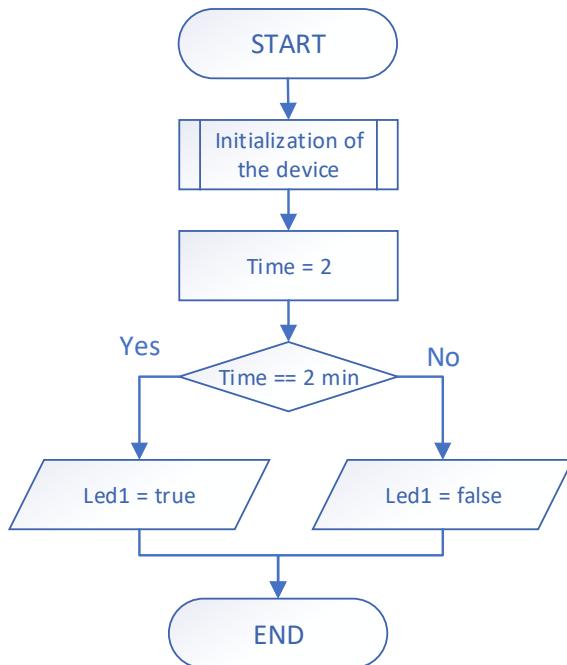


### Графични символи за изграждането на блокови схеми на алгоритми

В зависимост от броя на стъпките, които се извършват по време на изпълнението им, алгоритмите биват:

- **Линеен** – това е най-простият вид алгоритъм, при който броят на действията е равен на броя на стъпките, описани в него. Освен това действията от всяка отделна стъпка се извършват последователно и по реда на тяхното описание.
- **Разклонен** – това е алгоритъм, в който са поставени условия за разклонение на стъпките. При достигането на тези разклонения се прави избор за следващото изпълнение на стъпките от две или повече различни алтернативни действия. За реализирането на такъв алгоритъм броя на изпълняваните действия обикновено е различен от броя на описаните стъпки. В някои случаи действията могат да са

много малко на брой и бързо да се премине към край на алгоритъма. В други случаи изпълнението може да премине през стъпки, чийто брой е възможно най-голям от общия списък на алгоритъма. На фигурата по-долу е показана схема на разклонен алгоритъм.



*Схема на разклонен алгоритъм*

- **Цикличен** - това е алгоритъм, при който някои действия, нуждаещи се от многократно повторение, са организирани като циклично повтарящ се блок, докато се достигне до величина удовлетворяваща проверяваното условието. При този вид алгоритми, извършените действия надвишават многократно броя на отделните стъпки от списъка. На фигурата по-долу е показана схема на цикличен алгоритъм.

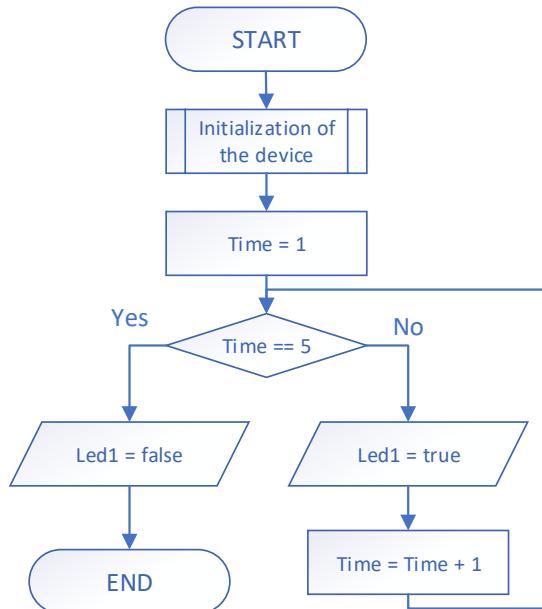


Схема на цикличен алгоритъм

- Последователен** - това е алгоритъм, при който действията се изпълняват едно след друго във времето (всяка стъпка се извършва в различен отрезък от времето).
- Паралелен** - това е алгоритъм, при който описаните действия в стъпките са независими едно от друго и позволяват възможността тези действия да се изпълняват паралелно (в един и същи отрезък от време).

#### ОЦЕНКА НА ЗНАНИЯТА

1. В зависимост от броя на изпълняваните стъпки, какви биват алгоритмите?

- цикличен
- линеен
- последователен
- алгоритмичен
- кръгов
- разклонен

2. В зависимост от времето, в което се изпълняват, какви биват алгоритмите?

- кръгов
- последователен
- алгоритмичен
- паралелен
- разклонен

Верни отговори: 1 (a, b, e); 2 (b, c)



## Разклоняващи структури в програмите с оператори switch .. case

При писането на програми, понякога се налага необходимостта да се извърши многовариантно разклонение. То би могло да се получи и чрез структура изградена от многократно повтарящи се оператори **if .. else**. Например, ако трябва да се провери състоянието на дадена променлива в границите от 0 до 100, то това означава че трябва да се опишат толкова блока с операторите **if.. else**. Това не е толкова смислено и удобно и затова се използва структура с друга двойка оператори **switch .. case**. На фиг. 1 е показан синтаксисът на този вид многовариантно разклонение.

```

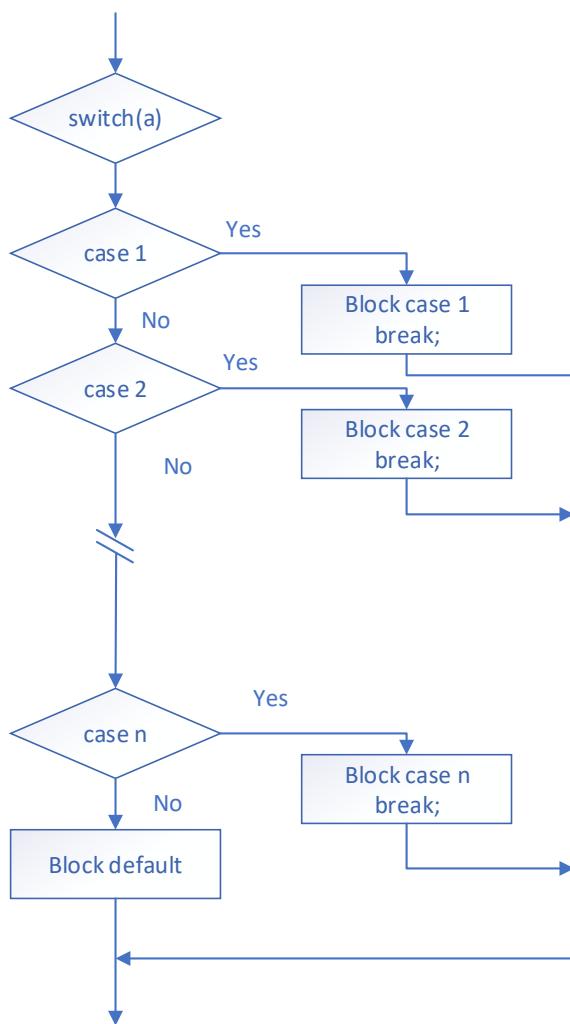
136  switch(variable) {
137      case 1: {
138          ...
139          break;
140      }
141      ...
142      ...
143      case 99: {
144          ...
145          break;
146      }
147      default: {
148          ...
149      }
150 }
```

Фиг. 1: Синтаксис на многовариантно разклонение с оператори **switch .. case**

При използването на оператори **switch .. case**, в зависимост от стойността на променливата се изпълнява един или друг блок с програмен код. Във всички блокове на **case** може да се опишат всякакви операции. Операторът **break**, указва завършване на съответния блок **case** и препраща управлението на програмата извън конструкцията **switch .. case** (след ред 150 от фиг. 1). Ако операторът **break** не бъде поставен в края на блок **case**, то тогава изпълнението на програмата продължава и в следващия блок **case**, докато не достигне до някакъв оператор **break**. Това е полезно в случаите, когато има необходимост да се изпълнява кодът от няколко поредни блока на оператора **case**.

Кодът, поставен в блока на оператора **default** се изпълнява само в случаите, когато стойността в променливата, проверявана от оператора **switch** не е описана в условието на нито един от блоковете **case**.

На фиг. 2 е показана блокова схема на алгоритъм за многовариантно разклонение с операторите **switch .. case**.



Фиг. 2: Блокова схема на алгоритъм за многовариантно разклонение с оператори `switch .. case`

## Видове програмни езици

В днешно време, програмистите разполагат с голям набор от различни програмни езици. Наличието на толкова голям брой програмни езици, дава възможност за тяхната класификация по различни признаки. Основно те се делят на два вида, езици от ниско ниво и такива от високо ниво.

Езиците от ниско ниво са:

- **Машинни езици.** Това са програмни езици, които са трудно разбираеми за хората. Същественото при тях е, че те са специфични и са конкретно ориентирани към точно определен вид процесор, т.е. всеки процесор може да изпълнява програми написани на собствения му машинен език. Това означава, че програмист работещ с даден вид микроконтролер, не може да програмира контролер който работи с друг машинен език.
- **Асемблерни езици.** Тези езици са близки до машинния, като при тях вместо двоичен код операндите се записват с думи наречени мнемоничен код.

Езиците от високо ниво са:

- **Процедурно ориентирани езици.** В тези езици, с помощта на ключови думи (взети от човешкия език) с точно определен синтаксис, се извършва описането на всички необходими действия, които процесорът трябва последователно да извърши. След това програмата се компилира до машинен език за съответния процесор. Всяка една от ключовите думи на езика от високо ниво, се описва с няколко инструкции на машинен език. Това води до повишаване на производителността при писането на програмата. В същото време компилирането води до намаляване бързодействието на програмата при нейното изпълнение, в сравнение с такава, написана на машинен език. Към тези езици се отнасят Basic, Pascal, C и др.
- **Обектно ориентирани езици.** Това са езици от най-високо ниво, които са разработени на базата на процедурно ориентираните езици. При тях могат да се използват готови библиотеки с предварително разработени класове и компоненти. Също така, има възможност от наследяване на свойства и параметри на съществуващите класове. Освен това, те дават възможност на програмиста да създава свои собствени обекти и класове. Към тези езици се отнасят Visual Basic, Turbo Pascal, C++, Delphi, CBuilder, Java и др.

Езиците за високо ниво се нуждаят от допълнителни програми, с които да може написаната програма с обикновен текстов редактор, да се преведе на машинен език и след това да бъде изпълнена от съответния процесор. Тези помощни програми са:

- Транслатори – това са програми, чрез които описаните в програмата текстови оператори се транслират (превеждат) на езика на процесора (машинен, двоичен код). Всеки език има свой транслатор и то за всеки отделен процесор. От своя страна, като начин на действие, транслаторите биват два вида:
  - Компилатори – при тях изпълнението на програмата се изпълнява на два етапа. В първия, програмата се превежда до изпълним код и се запазва на носител. При втория, се стартират и изпълняват преведените и записани на носител, машинни инструкции.
  - Интерпретатори – чрез тях се постига последователно изпълнение на двета етапа: компилиране и изпълнение. Компилираният код се запазва в RAM паметта и след това се стартира. При всяко ново стартиране на програмата се изпълняват всеки път и двета етапа.



## Изпълнителни устройства и задвижвания

### ВИДОВЕ ЕЛЕКТРИЧЕСКИ ЗАДВИЖВАЩИ УСТРОЙСТВА

Електродвигателят е електромеханично устройство, което с помощта на двуполюсен магнит преобразува електрическата енергия в механична. Електродвигателите се употребяват най-много като устройства за задвижване. Общото между всички електродвигатели е, че те се състоят от една неподвижна и една подвижна част. В неподвижната част (най-често това е корпуса на двигателя, наречен още статор) се поставя източник на насочен магнитен поток. Подвижната част най-често е вал, на който има навит проводник, по който протича електрически ток. Електрическият ток, който протича през проводника в ротора, поставен в магнитното поле на статора, генерира отклоняваща го сила, която задвижва механично вала (ротора) на двигателя.

В зависимост от конструкцията им и електричеството, което използват електродвигателите биват няколко вида.

- **Постояннотокови електродвигатели (ПТД)**

За да извършват никаква работа тези двигатели използват постоянен ток (прав ток). В статора на този вид двигатели са вградени магнитни полюси, които осигуряват постоянно магнитно поле. Полюсите могат да бъдат изработени от бобини с навит проводник върху сърцевина или чрез постоянно магнитни за създаване на хомогенно постоянно магнитно поле. При подаване на електричество, бобините осигуряват необходимото постоянно магнитно поле.

От своя страна ПТД могат да се разделят според типа на възбуджащите намотки на статора на:

- ПТД с отделно (независимо) възбуждане. При тях възбуджащите намотки на статора се захранват от отделен източник на ток;
- ПТД с последователно възбуждане. При този тип, възбуджащите намотки на статора са свързани към главното захранване, последователно на веригата на ротора;
- ПТД с паралелно възбуждане. При тези електродвигатели, възбуджащите намотки на статора са паралелно свързани към веригата на ротора.

В миналото ПТД са били широко разпространени и са се използвали почти навсякъде. За съжаление обаче при тях има съществен недостатък. За да протича ток през ротора на двигателя се използват четки от графит, които механично се тряят върху колектора на ротора. Това механично трение води до износване, а от там и до чести повреди. Освен това при тях има необходимост от предварително изправяне на тока, който те консумират. И не на последно място, те са доста по-скъпи за производство.

- **Променливотокови електродвигатели**

Този тип електродвигатели използват като източник на енергия променлив ток. С развитието на технологиите управлението на променливотоковите електродвигатели става все по-лесно и по-ефективно в сравнение с ПТД. Според вида на електрическото напрежение, което използват, променливотоковите двигатели могат да бъдат еднофазни или трифазни. И двата вида електродвигатели са намерили доста широко приложение както в бита така и в индустрията. Еднофазните се използват най-вече в бита, докато

трифазните се употребяват много повече в промишлеността, поради по-голямата мощност, която могат да отдават.

Според конструкцията си променливотоковите електродвигатели се разделят на два вида - асинхронни и синхронни.

- ***Асинхронни електродвигатели***

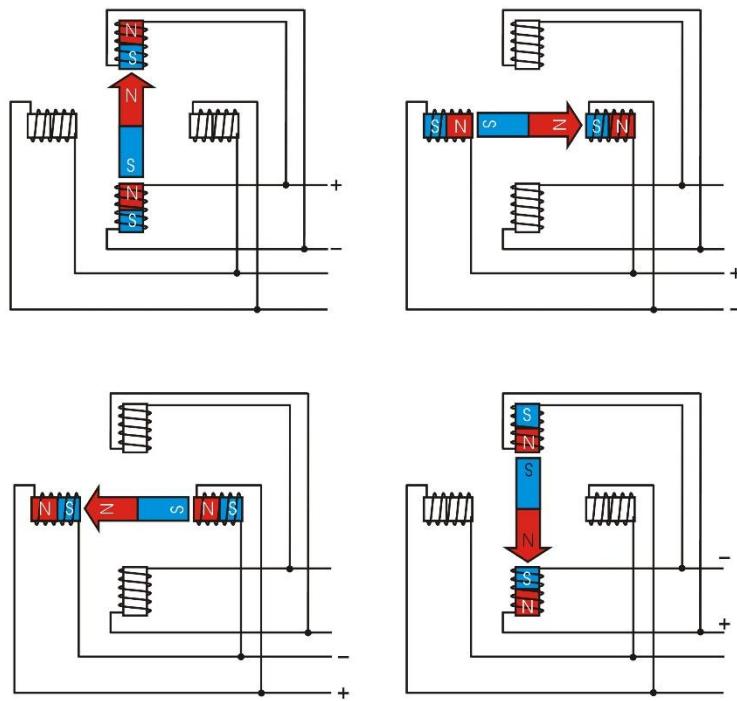
Те са най-разпространеният вид електродвигатели. Това се дължи на качествата, които притежават - ниска цена, опростена конструкция, лесни за производство, липса на механично износващи се детайли и др. Както и ПТД, асинхронните двигатели се състоят от неподвижна част (статор) и подвижна (ротор). В статора се генерира въртящо се магнитно поле с определена честота. В ротора са вградени така наречените навивки накъсо. Въртящото се магнитно поле на статора, създава електродвижещо напрежение в намотките на ротора и така го увелича в същата посока на въртене. Специфичното при работата на този вид двигатели е, че оборотите на ротора се приближават до оборотите на въртящото се магнитно поле, но винаги леко изостават в зависимост от натоварването на двигателя и никога не успяват да ги достигнат (роторът се движи асинхронно спрямо магнитното поле). Това е така нареченият ефект на хълзгане.

- ***Синхронни електродвигатели***

Принципът на работа на тези електродвигатели е сходен с този на асинхронните електродвигатели. При тях роторът се върти точно с честота на въртящото се магнитно поле в статора (синхронно). Синхронните електродвигатели са със сравнително ниска честота на въртене на ротора, която е в границите от 100 до 1000 об./мин. Използват се обикновено там, където се изисква постоянна честота на оборотите на ротора, независимо от натоварването. Също така са подходящи и се използват най-вече като генератори. Недостатък на синхронните електродвигатели е, че въртящият им момент при начално развъртане е много малък и е максимален само при достигане на номиналните си обороти на въртене. Пусковият им въртящ момент е равен на нула. Поради този недостатък, стартирането на синхронен двигател директно от захранващата мрежа създава проблеми. Това се извършва с помощта на външен източник за развъртане или от специална допълнителна намотка за стартиране (така нареченото асинхронно пускане). Тъй като оборотите на електродвигателя са синхронни на въртящото се магнитното поле, то за управлението им е много удачно да се използва промяна на честотата на това магнитното поле.

- ***Стъпкови електродвигатели***

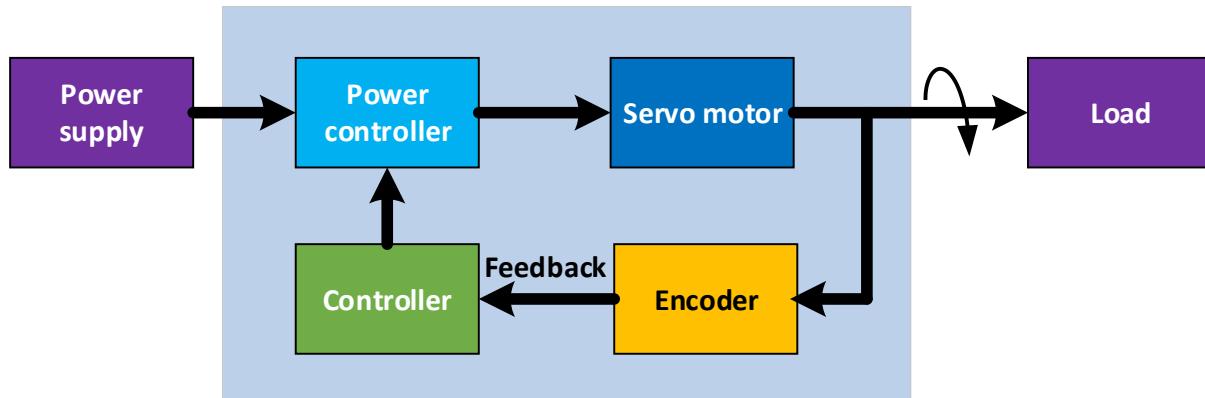
Движението при тези двигатели се извършва на стъпки (импулси). Статорните навивки са разположени като двойка полюси, като едната от тях се захранва в права посока, а другата в обратна. Докато в едната се образува привличащо магнитно поле, то в срещуположната се образува отблъскващо магнитно поле. Така при всяка отделна стъпка роторът се установява в постоянна позиция спрямо статора. При бързо редуване на следващите съседни навивки се получава въртящо се магнитно поле, което се върти около статора на определени стъпки. Стъпковите двигатели се използват там, където не се изисква голяма мощност при задвижването на различните механизми. Предимство на стъпковите двигатели е, че движението на ротора може да бъде спирано в точно определена позиция и пускано отново. За да се постигне по-точно позициониране е необходимо увеличаване броя на двойката полюси в статора. Недостатък е, че за управление на движението на стъпковия двигател е необходима специализирана схема. На фигурата по-долу е показана принципна схема на стъпков електродвигател.



Принципна схема на стъпков електродвигател

- **Сервозадвижване.**

Сервозадвижването е сложна система, състояща се от няколко основни модула: източник на електроенергия, изправител на напрежение, управляващ контролер, силов контролер, серводвигател и енкодер (фиг. 5.2). Управлението на вала на двигателя става с изключителна прецизност. За целта се извършва управление и следене на достигнатите стойности по три основни параметъра спрямо вала на двигателя - скорост, позиция и момент на натоварване. За източник на енергия най-често се използва напрежението от електрическата мрежа 220V. Полученото напрежение се изправя от изправителния блок и се подава към силовия контролер. Управляващият контролер изработва управляващи сигнали в зависимост от заданието, чрез които управлява силовия контролер. Силовият контролер изработва напрежения с необходимата честота и амплитуда и ги подава към серводвигателя. Към вала на серводвигателя е свързан прецизен абсолютен енкодер, който чрез обратната връзка подава информация за положението на вала във всеки един момент. Благодарение на тази информация управляващият контролер сравнява зададените стойности, изработените управляващи сигнали и достигнатото положение на вала. При необходимост, следващите управляващи сигнали се изработват с цел корекция за постигане на точно позициониране на вала.



Структура на сервоздвижване

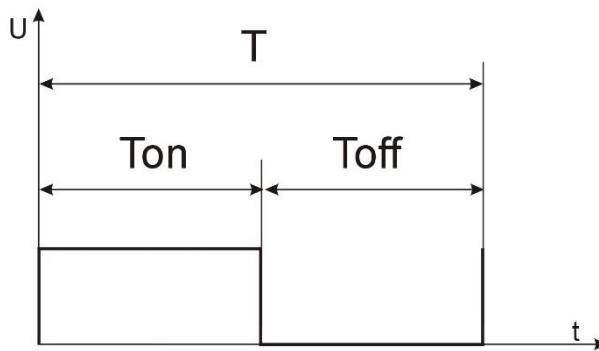
При сервоздвижващите системи най-същественото е определянето на позицията на вала във всеки един момент и обвързването му в обратна връзка с управлението на двигателя. При по-сложните системи позицията се определя от прецизен енкодер с достатъчно добра разделителност (достигаща до 10 000 точки за  $360^{\circ}$ ).

В днешно време, на пазара могат да се намерят сервоздвижвания, които са сравнително евтини и напълно задоволяват изискванията за конструиране на любителска роботизирана система. Едно такова сервоздвижване е sg90 (виж фигурата по-долу).



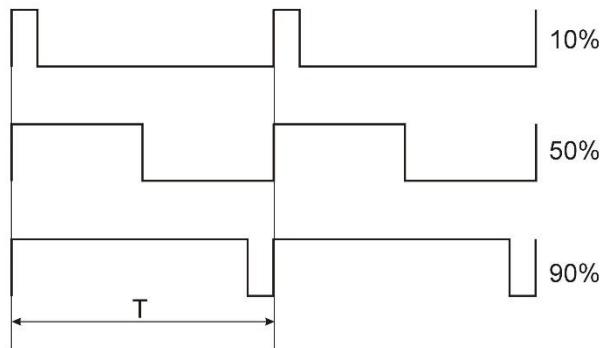
Система за сервоздвижване sg90

За работа на серводвигателя, показан на фигурата по-горе, е достатъчно да се свържат трите проводника. Два от тях са за подаване на захранващо напрежение (в случая 5V). Третият проводник служи за подаване на управляващ сигнал. Управлението на този вид серводвигател се извършва чрез така наречената широчинно-импулсна модулация (ШИМ). На фигурата по-долу е показан графичен вид на един период от ШИМ сигнал.



Един период на шифочинно-импулсно модулиран сигнал

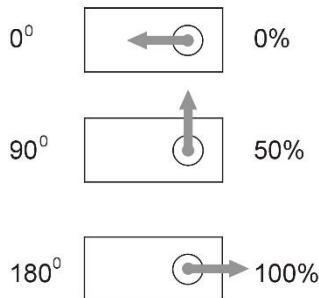
Това е импулсен сигнал с определена честота като запълването на даден период може да се променя в границите от 0 до 100%. На фигурата по-долу са показани ШИМ сигнали с различно запълване на периода.



ШИМ сигнали с различно запълване на периода

Периодът се изразява във време. Времето за което протича един период е  $T = 1/f$ , където  $f$  е честотата.

Управлението на серводвигателя SG90 се извършва като се променя запълването на периода на управляващия сигнал. В зависимост от запълването на сигнала, серводвигателят са завърта на съответния ъгъл от 0 до 180°. На фигурата по-долу са показани различни ъгли на завъртане на двигателя в зависимост от процентното запълване на ШИМ сигнала.



Различни ъгли на завъртане на двигателя в зависимост от процентното запълване на ШИМ сигнала

Подкрепата на Европейската комисия за изготвянето на настоящата публикация не представлява одобрение на съдържанието, което отразява гледните точки само на авторите и не може да се търси отговорност от Комисията за всяка употреба, която може да бъде използвана за информацията, съдържаща се в нея.



### ОЦЕНКА НА ЗНАНИЯТА

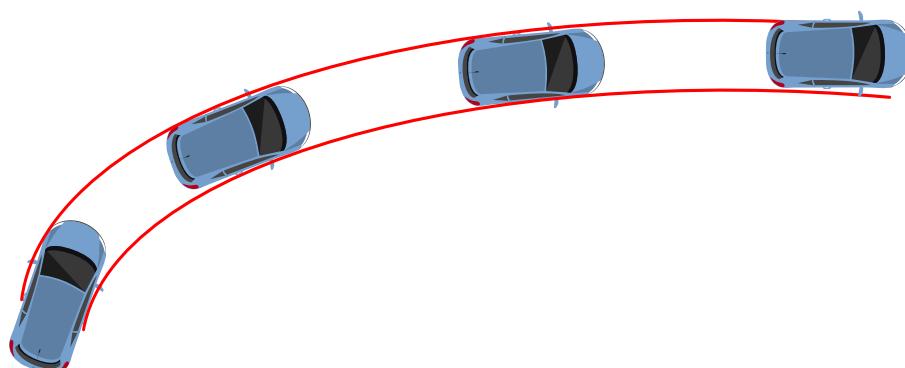
1. Кои от изброените електродвигатели работят с напрежение директно от захранващата мрежа?
  - a) правотоков
  - б) стъпков
  - в) синхронен
  - г) серводвигател
  - д) асинхронен
2. Кой от изброените компоненти влиза в състава на двигател със сервозадвижване.
  - а) управляващ контролер
  - б) енкодер
  - г) военната област
  - д) изправител на напрежение
3. Какви са границите на запълване на един период от ШИМ?
  - а) от 0 до 180%
  - б) от 0 до  $180^\circ$
  - в) от 0 до  $100^\circ$
  - г) от 0 до  $90^\circ$
  - д) от 0 до 100%
4. На колко градуса ще се завърти оста на серводвигател SG90, при подаден ШИМ управляващ сигнал с 50% запълване?
  - а)  $360^\circ$
  - б)  $60^\circ$
  - в)  $180^\circ$
  - г)  $90^\circ$
  - д)  $100^\circ$
5. При захранващо напрежение 12V, какво ще бъде сумарното напрежение на ШИМ сигнал с 25% запълване?
  - а) 12V
  - б) 6V
  - в) 24V
  - г) 10V
  - д) 3V

Верни отговори: 1 (в, д); 2 (а, б, д); 3 (д); 4 (г); 5 (д)



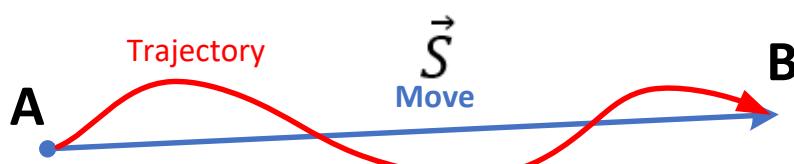
## Физически характеристики на механичното движение

Движението е една от най-важните задачи за робота. От физическа гледна точка механичното движение е промяна в положението на твърдото тяло в пространството спрямо другите. Твърдото тяло е система от материални точки, разстоянията между които са постоянни. Основните видове движения на твърдо тяло са трансляционни и ротационни. Трансляционното движение е движение, при което всички точки на тялото имат една и съща траектория (виж фигурата по-долу). Ротационното движение е движение, при което траекториите на различни точки на тялото са окръжности (или дъги от окръжности) с обща ос.



Трансляционно движение

Механичното движение се характеризира с физичните величини скорост, ускорение, изминато разстояние, изместване, траектория. Траекторията е линията, по която тялото се движи. По вид на движението траекториите се разделят на праволинейни и криволинейни. Изминатият път е дължината на траекторията на движещо се тяло, измерена в метри. Преместването е вектор, свързващ началните и последващите позиции на тялото. Траекторията и преместването се измерват в метри. Разликата между тях е показана на фигурата по-долу.



Разлика между понятията "преместване" и "траектория"

Скоростта характеризира бързината на промяната в положението на тялото, изчислена като съотношението на преместването към периода от време, през който е извършено. Ускорението характеризира бързината на промяната в скоростта на тялото. То се изчислява като съотношението на промяната в скоростта към времевия интервал, през който е била извършена промяната.

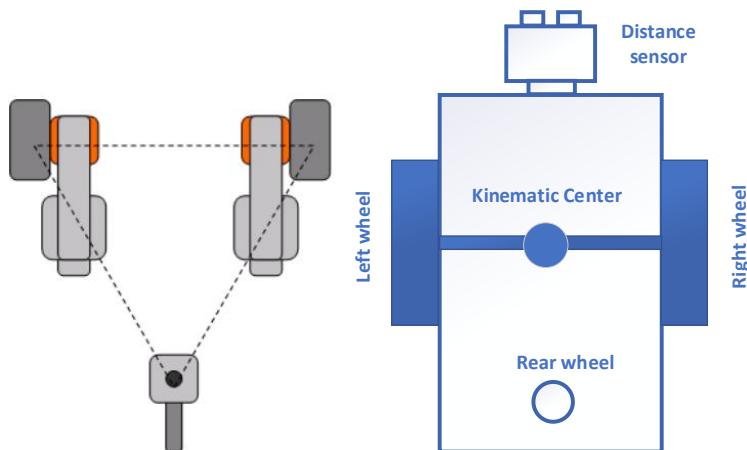
При движение роботът (двигателите на робота) извършва работа. В резултат от тази работа се преодоляват силите на триене и роторите на двигателите ще се завъртят. Всяко механично устройство се характеризира с такава физическа величина като мощност.

Мощността е физична величина и представлява отношението на пренесената енергия (или работата извършена от дадена сила) за определен интервал от време към големината на този интервал. Значително може да се увеличи или намали мощността на работа като се използват механични предавки.

### КИНЕМАТИКА НА УЧЕБЕН МОБИЛЕН РОБОТ

Кинематиката е част от механиката, която изучава математическото описание на движението на идеализираните тела (материална точка, абсолютно твърдо тяло, идеална течност), без да се вземат предвид причините за движението (маса, сили и т.н.). Кинематиката е един от основните етапи на изследванията при проектирането на мобилни роботи. Резултатът от кинематичния анализ е математическо описание на поведението на механичната система за по-нататъшното развитие на програмното управление на движението на робота.

Нека разгледаме модела на робот под формата на двувдигателна количка, базирана на конструктора Lego Mindstorms EV3 (виж фигурата по-долу). Това е най-често срещаният вид колесен робот. Количката може да бъде с три опорни точки, две от които са задвижващите колела, а третата е рулевото или свободно въртящото се колело. Центърът на тежестта не трябва да се намира над рулевото колело, а по-близо до водещите колела. Работът е моделиран като твърдо тяло на колела, движещи се по хоризонтална равнина.



Модел на двуколесен робот

За да се напише програма за движение на този робот, трябва да се укаже продължителността на работа на двигателите, която може да бъде зададена в градуси, обороти или секунди. Нека да разгледаме едно примерно задание за движение на робота: движение направо на 50 см и завъртане под ъгъл  $90^\circ$ . Важният параметър - **продължителност на работата на двигателите** се определя по метода на пробата и грешката или се изчислява по формула, получена на базата на кинематичен анализ. За да се осигури висока точност на движение на робота, ще изчислим продължителността на работа на двигателите в градуси. Нека например, при един оборот на колелото, роботът прави завой на  $360^\circ$ , т.e.  $2 \cdot \pi \cdot r = 360^\circ$  ( $r$  е радиусът на колелото).

Продължителността на работа на двигателите (в градуси)  $\varphi_{лев}$  и  $\varphi_{дес}$  се изчислява по формулата съгласно схемата на фигурата по-долу, където  $s$  е разстоянието, на което роботът трябва да се премести (в нашия случай  $s = 50 \text{ cm}$ ),  $r$  е радиусът на колелото ( $r = 2,8 \text{ cm}$  за разглеждания модел на робот). В резултат на това получаваме  $\varphi_{лев} = \varphi_{дес} = 1023^\circ$ .

$$\varphi_{десен} = \varphi_{ляв} = s \cdot \frac{360^\circ}{2\pi r}$$

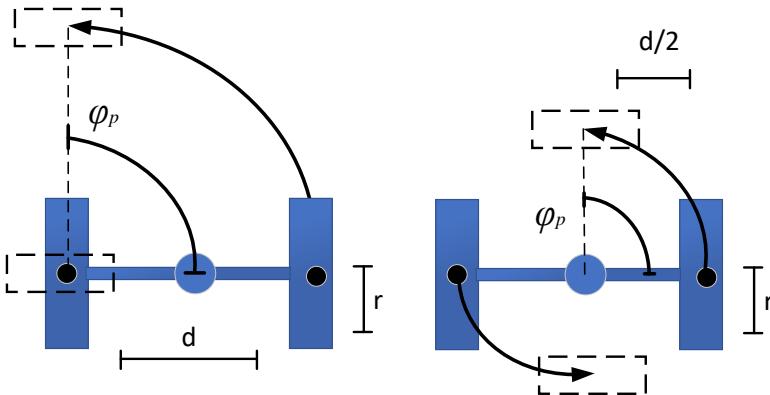


Схема на въртене на робота около лявото колело и около центъра на робота

За да може роботът да се завърти на  $90^\circ$  ( $\varphi_p = 90^\circ$ ) наляво, трябва да знаем  $d$  - разстоянието между осите на колелата на робота ( $d = 11,5$  см за разглеждания модел на робот),  $r$  е радиусът на колелото ( $r = 2,8$  см). Ако роботът се върти около лявото колело, тогава  $\varphi_{ляв} = 0$ , а  $\varphi_{десен}$  се изчислява по формулата:

$$\varphi_{десен} = \varphi_p \cdot \frac{d}{r}$$

В резултат на изчисленията се получава продължителността на работа на десния двигател  $370^\circ$ .

**Ротацията на робота около неговия център** (завъртане на място) се извършва по такъв начин, че десният двигател работи в посока напред, а левият двигател се върти в обратна посока, а продължителността на работа на двигателите ще бъде еднаква. Според схемата, представена на по-горе, продължителността на работа на двигателите се изчислява по формулите:

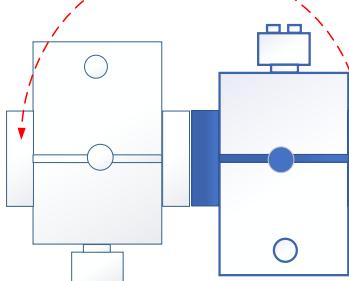
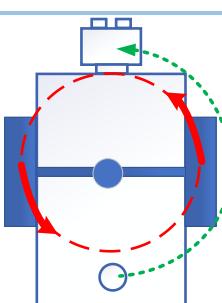
$$\varphi_{десен} = \varphi_p \cdot \frac{d/2}{r}; \varphi_{ляв} = -\varphi_p \cdot \frac{d/2}{r}$$

В резултат на изчисленията получаваме  $\varphi_{десен} = 188^\circ$ ,  $\varphi_{ляв} = -188^\circ$ . Знакът "-" означава, че левият двигател ще бъде програмиран да се върти "назад".

Математически получените стойности за продължителността на въртене на двигателите не осигуряват максимална точност на движението на робота. Необходимо е да се вземе предвид инерцията на двигателите, силите на триене, които възникват между колелата на робота и повърхността, върху която се движи, нивото на зареждане на акумулаторната батерия на модула и т.н.

Следователно, като правило, получените стойности се коригират спрямо конкретните зададени условия за робота. Изборът на метод за въртене около едно от колелата или на място зависи от естеството на решавания проблем. Ако роботът направи завой около едното колело, тогава точността на такъв завой е много по-голяма, като в същото време отнема много повече място за маневриране, а скоростта му е много по-ниска, отколкото при завъртане на място.

Таблица 1: Сравнение на начините за извършване на въртене на двумоторна количка

Завъртане около едното колело	Завъртане на място
	
$\varphi_{\text{десен}} = \varphi_p \cdot \frac{d}{r};$ $\varphi_{\text{ляв}} = 0$	$\varphi_{\text{десен}} = \varphi_p \cdot \frac{d/2}{r};$ $\varphi_{\text{ляв}} = -\varphi_p \cdot \frac{d/2}{r}$
<b>Пространство на завъртане</b>	
ГОЛЯМО	МАЛКО
<b>Скорост на завъртане</b>	
МАЛКА	ГОЛЯМА
<b>Точност на завъртане</b>	
ГОЛЯМА	МАЛКА

### ДВИЖЕНИЕ НА РОБОТА ПО ПРАВА ЛИНИЯ

Един ход на задвижващия механизъм на серводвигателя е 360 градуса. За да може роботът да измине разстоянието, което е необходимо, се изчислява броят на градусите, на които роторът трябва да се завърти. Използва се следната формула:

$$\varphi_p = \frac{S}{L} 360^\circ$$

където:  $\varphi_p$  – ъгъл на завъртане,  $S$  – изминато разстояние;  $L$  – периметър (обиколка) на колелото.

Например, да определим на какъв ъгъл трябва да се завърти серводвигателя за да може роботът да измине разстояние от 500 mm при положение, че радиусът на колелата е  $r = 28 \text{ mm}$ . От математиката знаем, че обиколката на окръжност се изчислява по формулата:

$$L = 2\pi r = 2 \cdot 3,14 \cdot 28 = 175,84 \text{ mm}$$

Следователно, ъгълът на завъртане ще бъде:

$$\varphi_p = \frac{S}{L} 360^\circ = \frac{500 \text{ mm}}{175,84 \text{ mm}} 360^\circ = 1023.66^\circ$$



По този начин, общият брой на степените, до които задвижващият механизъм на серводвигателя трябва да се завърти, за да се изминат 50 см от пътя е 1023,66 mm.

### ВЪПРОСИ ЗА САМОПРОВЕРКА

1. Механичното движение може да бъде:

- a) праволинейно или криволинейно
- б) равномерно или неравномерно
- в) равномерно ускорено или на равномерно забавяще
- г) динамични и импулсно

2. Ротацията на робота около неговия център в лява посока се извършва в съответствие с формулите:

- a)  $\varphi_{\text{десен}} = -\varphi_p \cdot \frac{d/2}{r}$ ;  $\varphi_{\text{ляв}} = \varphi_p \cdot \frac{d/2}{r}$
- б)  $\varphi_{\text{десен}} = \varphi_p \cdot \frac{d/2}{r}$ ;  $\varphi_{\text{ляв}} = -\varphi_p \cdot \frac{d/2}{r}$
- в)  $\varphi_{\text{десен}} = 0$ ;  $\varphi_{\text{ляв}} = -\varphi_p \cdot \frac{d/2}{r}$
- г)  $\varphi_{\text{десен}} = \varphi_p \cdot \frac{d/2}{r}$ ;  $\varphi_{\text{ляв}} = 0$

3. Ротацията на робота около неговия център в дясна посока се извършва в съответствие с формулите:

- a)  $\varphi_{\text{десен}} = -\varphi_p \cdot \frac{d/2}{r}$ ;  $\varphi_{\text{ляв}} = \varphi_p \cdot \frac{d/2}{r}$
- б)  $\varphi_{\text{десен}} = \varphi_p \cdot \frac{d/2}{r}$ ;  $\varphi_{\text{ляв}} = -\varphi_p \cdot \frac{d/2}{r}$
- в)  $\varphi_{\text{десен}} = 0$ ;  $\varphi_{\text{ляв}} = -\varphi_p \cdot \frac{d/2}{r}$
- г)  $\varphi_{\text{десен}} = \varphi_p \cdot \frac{d/2}{r}$ ;  $\varphi_{\text{ляв}} = 0$

4. Ротацията на робота около дясното колело се извършва в съответствие с формулите:

- a)  $\varphi_{\text{десен}} = -\varphi_p \cdot \frac{d/2}{r}$ ;  $\varphi_{\text{ляв}} = \varphi_p \cdot \frac{d/2}{r}$
- б)  $\varphi_{\text{десен}} = \varphi_p \cdot \frac{d/2}{r}$ ;  $\varphi_{\text{ляв}} = -\varphi_p \cdot \frac{d/2}{r}$
- в)  $\varphi_{\text{десен}} = 0$ ;  $\varphi_{\text{ляв}} = -\varphi_p \cdot \frac{d/2}{r}$
- г)  $\varphi_{\text{десен}} = \varphi_p \cdot \frac{d/2}{r}$ ;  $\varphi_{\text{ляв}} = 0$

5. Ротацията на робота около лявото колело се извършва в съответствие с формулите:

- a)  $\varphi_{\text{десен}} = -\varphi_p \cdot \frac{d/2}{r}$ ;  $\varphi_{\text{ляв}} = \varphi_p \cdot \frac{d/2}{r}$
- б)  $\varphi_{\text{десен}} = \varphi_p \cdot \frac{d/2}{r}$ ;  $\varphi_{\text{ляв}} = -\varphi_p \cdot \frac{d/2}{r}$
- в)  $\varphi_{\text{десен}} = 0$ ;  $\varphi_{\text{ляв}} = -\varphi_p \cdot \frac{d/2}{r}$



$$\text{г) } \varphi_{\text{десен}} = \varphi_p \cdot \frac{d/2}{r}; \varphi_{\text{ляв}} = 0$$

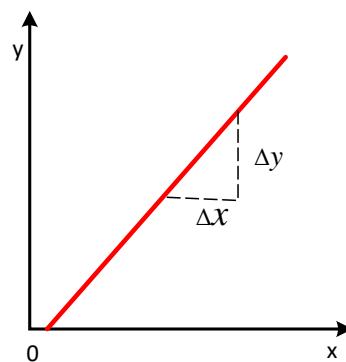
Верни отговори: 1 (а, б, в); 2 (б); 3 (а); 4 (в); 5 (г)



## Сензорът като чувствителен елемент в системата на робота

Тъй като всеки сензор е инструмент за измерване, е важно да се разберат понятията „точност“ и „грешка“ на измерванията. Точността е характеристика на качеството на измервателния уред. Грешката в измерването е количествена оценка на получения резултат от измерването, изразена в разликата между измерената и действителната стойност на физическа величина.

Сензорът като средство за измерване се характеризира с диапазон на измерване. Това е обхватът на стойностите на измерваната физическа величина, в рамките на който сензорът може да я преобразува в измервателен сигнал. Функционалната връзка между входните и изходните стойности се нарича функция на преобразуване на сензора, която може да бъде представена като формула, графика или таблица. Най-често сензорите имат линейна функция на преобразуване (виж фигурата по-долу).



*Линейна функция на преобразуване*

За описание на линейната функция на преобразуване  $y = f(x)$ , където  $x$  е входящата, а  $y$  е изходната стойност, са необходими началната стойност  $y_0$ , стойността на входа  $x$ , и относителният наклон на характеристиката  $s = \Delta y / \Delta x$ , наречен чувствителност на сензора.

Сензорната система на роботите обикновено възпроизвежда функциите на човешките сетивни органи (зрение, слух, мириз, допир и вкус). Сензорите могат да открият някои външни сигнали и да определят тяхната големина, което води до появата на пропорционален електрически сигнал на изхода им. Информацията, която се съдържа в сигнала, трябва да се чете и обработва от микроконтролера на робота. Като правило типовете сензорни устройства се различават според вида на въздействието, на което реагират - светлина, звук, топлина и др.

*Таблица 2: Предназначение на сензорите в Lego Mindstorms EV3*

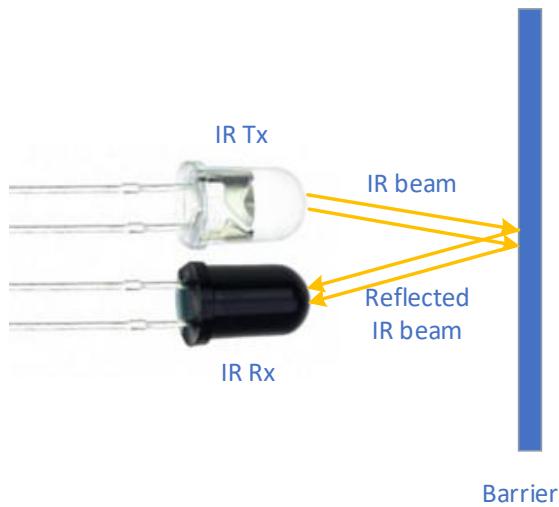
№	Сензор	Предназначение
1	Сензор за въртене на двигателя (енкодер)	Движение, преброяване на броя обороти на двигателя
2	Сензор за допир	Допир (съприкосновение)
3	Сензор за звук	Слух (отчита силата на звука)
4	Сензор за цвят/светлина	Зрение (определение на осветеността на помещението, яркостта на обектите, разпознаване на цветове)

5	Сензор за разстояние	Зрение (проследяване на препятствия, определяне на разстоянието до препятствията)
6	Инфрачервен сензор	Зрение (проследяване на препятствия, определяне на разстоянието до препятствията)
7	Жироскоп	Определяне на въртеливо движение
8	Сензор за ускорение (акселерометър)	Открива промяната на скоростта
9	Компас-сензор	Определяне на положението на тялото в пространството (ориентация по посоките на света)
10	Сензор за откриване на обекти	Сензорът за откриване на обекти дава възможност да се изгради робот, който може да открива обекти, използвайки отразени светлинни сигнали

## Следене на линия

### ПОНЯТИЯ ЗА LINE FOLLOWER

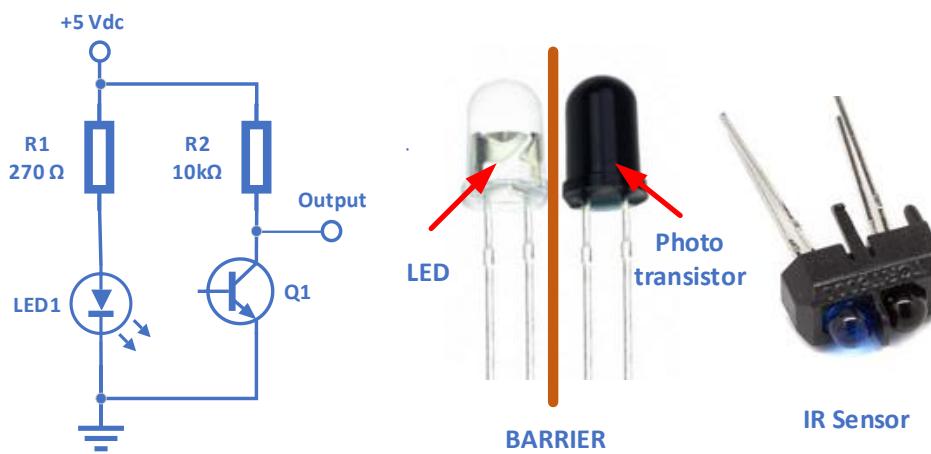
Концепцията за следене на линия от робот (Line Follower robot) е свързана със светлината. При нея се използва свойството абсорбция. Известно е, че черната повърхност погълща цялата светлина, която пада върху нея, а бялата я отразява. Това поведение на светлината се използва при следене на линия от роботите.



Функциониране на инфрачервен сензор

За използване на това свойство е необходим инфрачервен (IR) сензор. IR сензор се състои от излъчвател на инфрачервена светлина IR Tx и приемник IR Rx (виж фигурата по-горе). Излъчвателят представлява LED (Light Emitting Diode) диод, а приемникът може да бъде LDR (Light Dependent Resistor) или IRT (Infrared Photo Transistor).

Да предположим, че сензорът е разположен срещу бяла повърхност. Бялата повърхност отразява повече светлина в сравнение с всеки друг цвят. Ето защо повече светлина ще се отрази на IRT и той ще се отпуши (виж фигурата по-долу).



Базова схема на инфрачервен сензор за близост

Така на изхода на сензора ще се получи ниско ниво. Ако сензорът е разположен срещу черна повърхност, излъчената светлина от LED диода ще се погълне в по-голямата си

*Подкрепата на Европейската комисия за изготвянето на настоящата публикация не представява одобрение на съдържанието, което отразява гледните точки само на авторите и не може да се търси отговорност от Комисията за всяка употреба, която може да бъде използвана за информацията, съдържаща се в нея.*

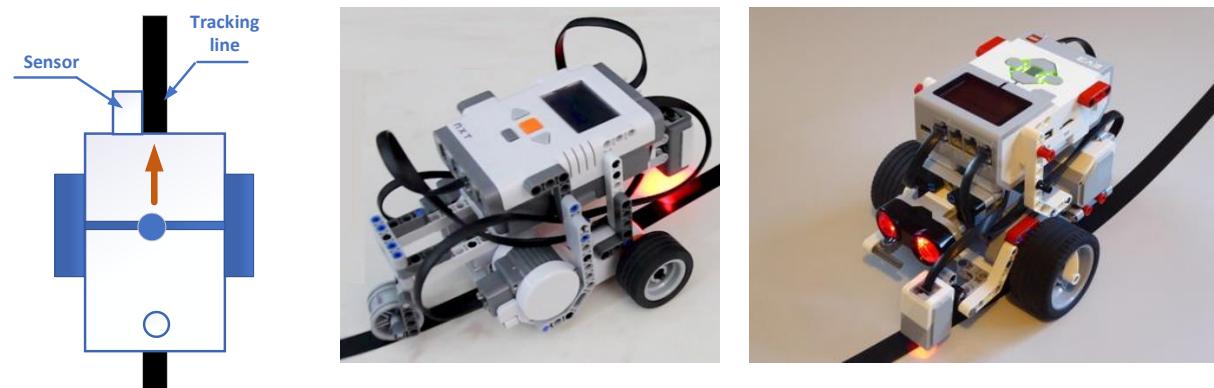
част и върху IRT няма да попадне достатъчно количество светлина за да се отпуши. Така на изхода ще се установи високо ниво. Тази разлика в отразената светлина помага да се определи дали сензорът е на линията или на околната повърхност. Използването на IR светлина намалява шума, дължащ се на околната светлина.

### АЛГОРИТМИ ЗА СЛЕДЕНЕ НА ЛИНИЯ

За да се движки роботът по траекторията, зададена от линия, е необходимо той да бъде снабден с един или няколко сензора за светлина. Колкото повече светлинни сензори има, толкова по-бързо роботът може да се движки, докато следва линията. В настоящия раздел се демонстрират алгоритми за следене на линия с един и два светлинни сензора.

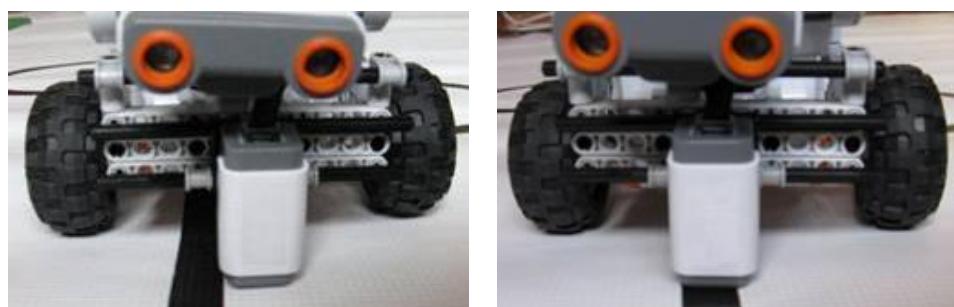
- **Базов алгоритъм за следене на линия (релеен регулатор) с един сензор**

За реализиране на алгоритъма за следене на линия по релеен закон е необходим двуколесен робот с едно опорно колело и сензор за цвят разположен в предната част на робота и насочен към пода (виж фигурата по-долу).



*Конструкция на робот за следене на линия с един сензор*

В първия разгledан случай няма да се следва линията, а ще се следва краят на черната линия. Като следваме границата на линията и ако сензорът за светлина “вижда бял цвят”, знаем, че роботът е в ляво от края на линията (виж фигурата по-долу). Ако “вижда черен цвят”, знаем, че той е отдясно на края на линията. Това се нарича “последовател на лявата линия”, тъй като той следва левия край на линията.

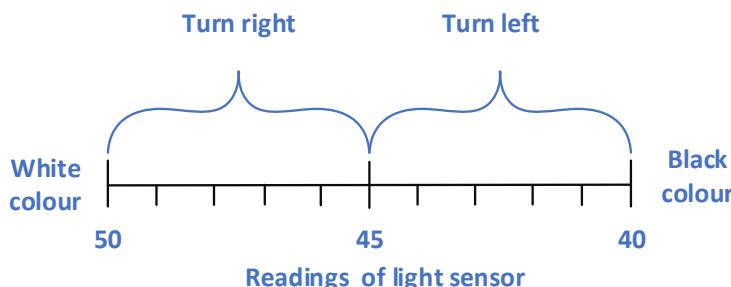


*Разположение на сензора спрямо линията, задаваща траекторията на движение*

За да се постигне движение в желаната посока се редува работата на левия или десния двигател, с което се прави придвижването напред по границата между черното и бялото.

Такъв алгоритъм се нарича „класически“ и с него започваме изучаването на движението по черна линия. Ако го разглеждаме като регулатор, то той се нарича релеен регулатор, т.e. контролът се извършва съгласно принципа на отворено /затворено (всеки момент от време и двата двигателя се редуват в две състояния - включване / изключване).

Трябва да знаем какви стойности връща светлинния сензор, когато “вижда бяло” и когато “вижда черно”. Един некалибриран сензор може да върне 50 при “бяло” и 40 при “черно” в скала от 0 до 100. Удобно е да се начертаят стойностите на една проста цифрова линия, за да се визуализира как преобразуваме стойностите на светлинните сензори в промени в движението на робота (виж фигурата по-долу).



Графика на алгоритъм за движение по линия при две зони

Ще разделим диапазона на две равни части и ще кажем, че ако нивото на светлината е помалко от 45, искаме роботът да завие наляво. Ако е по-голямо от 45, искаме да завие надясно. Например, ако показанието на сензора надвиши 45, това означава, че роботът е навлязъл в бялото поле. Тази ситуация налага левият двигател да работи, а десният да бъде спрян. В резултат на това, роботът ще завие на дясно. Когато стойността на сензора е помалка от 45 означава, че роботът се намира над черната лента и се налага да се направи корекция в движението като се спре левият двигател, а се включи десният. Така роботът ще завие наляво. Ето и един примерен код на този алгоритъм:

```
task main() {
    int grey=45; // the value of gray
    while (true) {
        if (SensorValue[S1]>grey) {
            motor[motorB]=100;
            motor[motorC]=0;
        } else {
            motor[motorB]=0;
            motor[motorC]=100;
        }
        wait1Msec(1);
    }
}
```

Този алгоритъм има редица недостатъци, които влияят върху качеството на изпълнението на задачата от робота (бързодействие и плавност при движението):

- във всеки един момент от времето един от двигателите не се върти;
- роботът може да пропусне черната линия поради високата скорост на двигателите му;



- в резултат на промяна в нивото на осветеност на помещението, роботът може да пропусне линията и да се завърти на място, което изисква допълнително отстраняване на грешки в програмата (задаване на праг на нивото на осветеност, подходящ за външните условия).

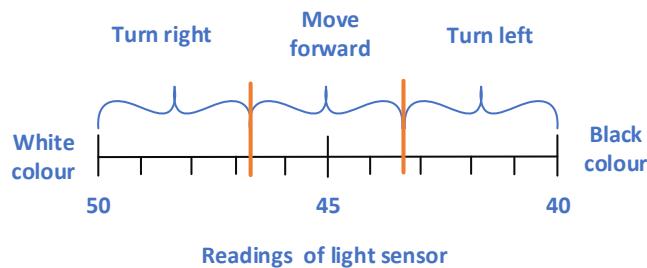
Тези недостатъци могат да се минимизират ако двигателите не се спират изцяло и не работят на пълна мощност. Така ще се повиши средната скорост но като цяло тя ще бъде доста ниска. Най-често възникващите проблеми в работата на алгоритъма са:

- роботът се върти на място без да влезе в линията. В този случай трябва да започнете от другата страна на линията или да промените свързването на двигателите към контролера на друг порт.
- Роботът пропуска линията, без да има време да реагира. Тук трябва да се намали мощността на двигателите.
- Роботът реагира на малък шум на бяло, без да достига черно. Необходимо е да се увеличи прагът на чувствителността на сензора (например не с 5, а с 8 точки).

Възможно е предварително да се определи нивото на осветеност в това поле и да се използва неговата абсолютна стойност. Използвайки показанията на сензорите на бяло и черно, получени чрез менюто View, изчисляваме тяхната средна аритметична стойност например  $(53 + 37) / 2 = 45$ , която ще наричаме „сива стойност“. Пресичайки стойността на сензора 45, роботът ще промени посоката на движението си. Очевидно е, че наляво от „сивото“ всички показания на сензора ще бъдат „бели“, а отдясно – „черни“.

- **Оптимизация на основния алгоритъм за следване на линията**

В посочения по-горе подход, роботът никога не се движи направо, дори ако е перфектно подравнен с края на линията и линията е права. Това не е ефективно и може да се подобри. За тази цел вместо да разделяме стойностите от сензора на две области, ще ги разделим на три. Тази идея се основава на това, че човекът и роботът не виждат по един и същи начин. Роботът не възприема цвета, а интензивността на отразената светлина, която попада върху фоточувствителния елемент на светлинния сензор. Следователно той е способен да причисли определени светлосенки към бялото или черното. Затова се предлага наличието на прагови стойности, над които или под които да не се прави корекция в движението на робота. Стойността на прага (средноаритметичното между показанията на сензорите на тъмна и светла повърхност) се възприема традиционно като „сиво“. Роботът определя светла повърхност когато получава от светлинния сензор стойности по-малки от прага (зададен в програмата), и тъмна повърхност когато получените стойности са над прага. Можете да изберете конкретен диапазон от стойности, които роботът възприема като „сив цвят“ (например между 43 и 47). В този случай роботът се движи направо, т.е. и двета двигателя работят в една и съща посока с еднаква мощност (виж фигурата по-долу).



### *Графика на алгоритъм за движение по линия при три зони*

Така, ако нивото на светлината е по-малко от 43, роботът ще завие наляво. Ако стойността на светлината е между 44 и 47, роботът ще се движи направо. Ако нивото на светлината е по-голямо от 47, роботът ще завие надясно. Тогава алгоритмът за следване на линията е малко по-сложен, но роботът ще премине по-бързо желаната траектория. Алгоритмът може лесно да се реализира с две проверки, както е показано в следния примерен код:

```
task main(){
    int grey_min=43; // minimum value of gray
    int grey_max=47; // maximum value of gray
    while (true) {
        if (SensorValue[S1]>grey_max)      {
            motor[motorB]=70;
            motor[motorC]=0;
        }else if (SensorValue[S1]<grey_min) {
            motor[motorB]=0;
            motor[motorC]=70;
        } else {
            motor[motorB]=100;
            motor[motorC]=100;
        }
        wait1Msec(1);
    }
}
```

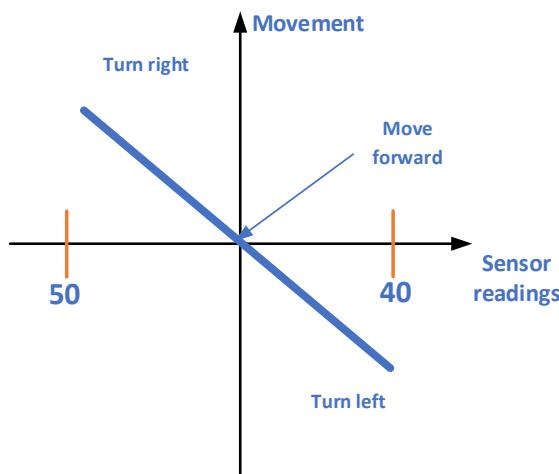
Този подход работи по-добре от първия. Сега роботът понякога се движи право напред. Както и при първия подход, вие трябва да решите каква скорост на въртене на двигателите да използвате, а това обикновено зависи от характеристиките на линията, която следвате.

- **Използване на ПИД-регулатор в алгоритми за следене на линия**

При първия подход, роботът може да направи само две неща - да завие наляво или надясно. Завоите са еднакви, но са в противоположни посоки. Във втория подход към двета завоя добавихме „върви направо“. Ако имаме повече от три диапазона, тогава се нуждаем от повече „видове“ завои.

За да разберем какви са тези „повече видове завои“, ще преобразуваме числовата линия в графика (виж фигураната по-долу).





Графика на алгоритъм за движение по линия при повече зони

На графиката хоризонталната ос съдържа показанията на светлинния сензор, а вертикалната ос - скоростта на въртене на двигателите. Ако показанието на светлинния сензор укаже, че сме близо до линията, тогава правим малък завой. Ако сме далеч от линията, ще направим голям завой т.е. скоростта на движение на двигателите е пропорционална на интензитета на отразения светлинен поток от повърхността.

ПИД-регулаторът (пропорционален интегрално-диференциален регулятор) е метод, широко използван за подобряване на работата на различни технически устройства. Математическото описание на този метод е доста сложно и се дава като правило в теорията на автоматичното управление. Регулирането е специален случай на управление.

Изходът на ПИД регулатора  $u(t)$  зависи от грешка на регулиране  $e(t)$ , от интеграла на тази грешка и от производната на грешката.

$$u(t) = K_p \cdot e(t) + K_I \int_0^t e(t) dt + K_D \frac{de(t)}{dt}$$

където  $K_p$  е коефициент на усиливане на пропорционалната част,  $K_I$  е коефициент на усиливане на интегралната част, а  $K_D$  е коефициент на усиливане на диференциалната част.

Пропорционалният компонент е отговорен за позиционирането на системата в дадено състояние. В някои случаи може да се предизвика пререгулиране с последващи автоколебания. Това означава, че П-регулаторът може да "прекали" и роботът ще започне да се движи от едната към другата страна (да криволичи).

Интегралният компонент натрупва отрицателен опит (обобщава грешките) и генерира компенсиращо въздействие. При минимални отклонения пропорционалният компонент „отслабва“, а интегралът, поради бързото му увеличаване чрез сумиране, помага да „изтегли“ регулираната стойност към зададената стойност.

Диференциалният компонент следи скоростта на промяна на състоянието на системата и предотвратява възникването на пререгулиране. В някои случаи той е противоположен по знак на пропорционалната съставяща, а при други съвпада по знак. Ако някои от компонентите не се използват, тогава контролерът се нарича пропорционално-интегрален (ПИ), пропорционално-диференциален (ПД), пропорционален (П). С помощта на ПИД регулатора ще подобrim оригиналния алгоритъм за движение по черната линия, описан по-горе. За тази цел се препоръчва отделно да се добави към алгоритъма всяка от трите



корекции (пропорционална, интегрална и диференциална) и да се отстрани грешката в работата на робота на всяка стъпка.

Пропорционален регулятор е устройство, което упражнява управляващо въздействие  $u(t)$  върху обект, пропорционално на неговото линейно отклонение  $e(t)$  от дадено състояние  $U_0(t)$ :

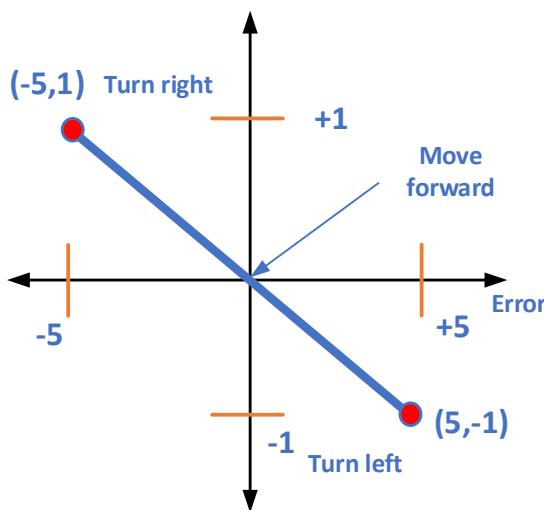
$$e(t) = U_0(t) - y(t)$$

където  $y(t)$  е състояние в даден момент;

$$u(t) = K_p \cdot e(t)$$

където  $K_p$  е коефициент на усилване (коффициент на пропорционалност).

В случая на класическия алгоритъм, само един от двигателите (ляв или десен) работи в даден момент. При използване на пропорционален регулятор, колкото повече роботът се отклонява от зададения курс, толкова по-активно двигателите трябва да работят, за да го коригират. На фигурата по-долу е представена графично идеята за пропорционален регулятор.



Графично представяне на идеята за пропорционален регулятор в алгоритъма за следене на линия

На оста X са показанията на светлинния сензор (от 40 до 50), които ще считаме за сиви. Стойността 45 е избрана като прагова, така че грешката ще бъде  $e(t) = \pm 5$ . Оста Y, минаваща през праговата стойност, указва границата на въртенето на оста на робота (ляво или дясно). Очевидно е, че скоростта на въртенето на робота е линейна функция в зависимост от показанията на светлинния сензор. Уравнението на линията се описва с израза  $y = k \cdot x + b$ , като в нашия случай  $y = k \cdot x$  ( $b = 0$ ). Коефициентът  $k$  е ъгъл на наклона на правата линия, изчислен по формулата  $k = \frac{dy}{dx}$ , а от друга страна това е коефициентът на пропорционалност (усилването) на регулятора. Стойността на този коефициент варира в границите от -1 до 1. Графиката, представена на фигурата по-горе отразява същността на пропорционалния регулятор. Грешката в интервала от -5 до 5 е разположена по оста X, а по оста Y - коефициентът на „усилване“ на въртенето в интервала от -1 (рязко наляво) до 1 (остъп десен завой). Стойността на коефициента  $k$ , който съвпада с коефициента на усилване в ПИД закона за регулиране  $K_p$  се определя по



следната зависимост като за изчисленията се вземат крайните точки на графиката  $(-5; 1)$  и  $(5; -1)$ :

$$K_p = k = \frac{dy}{dx} = \frac{\Delta y}{\Delta x} = \frac{y_{max} - y_{min}}{x_{max} - x_{min}} = \frac{(1 - (-1))}{(5 - (-5))} = \frac{2}{5} = 0.2$$

Както се вижда,  $k$  зависи от избрания диапазон на грешката, който е условно избран от нас. Извън диапазона от  $-5$  до  $+5$  не можем да кажем колко далеч е светлинният сензор от края на линията. Щом сензорът за светлина стане твърде далеч от края на линията, показанието на сензора вече не е пропорционално на грешката.

Обхватът, в който сензорът дава пропорционален отговор, се нарича "пропорционален обхват". В нашия случай аналогът на пропорционалния обхват на светлинния сензор е неговото отчитане в диапазона от  $40$  до  $50$  и грешката от  $-5$  до  $+5$ . За двигателите пропорционалният обхват е от  $-100$  (пълна мощност в обратната посока) до  $+100$  (пълна мощност при движение напред). Така пропорционалният регулятор работи ефективно само за малки ъгли на отклонение. Роботът трябва да бъде поставен в посоката на движението и за да се получи максимален ефект, сензорът трябва да е от лявата страна на черната линия.

В действителност стойността, връщана от П-регулатора представлява корекция на базовата скорост  $P_{ном}$  на движение на робота, като корекцията се добавя към скоростта на лявото колело и изважда от скоростта на дясното колело.

$$P_{лево} = P_{ном} + u(t)$$

$$P_{дясно} = P_{ном} - u(t)$$

Гореизложеното предполага, че пропорционалното управление на двигателите може да се извърши съгласно следния код:

```
task main() {
    const int grey_min = 43; // minimum value of gray
    const int grey_max = 47; // maximum value of gray
    const int grey = (grey_max + grey_min)/2; // the value of gray
    const float Kp = (1 - (-1)) / ((grey_min-grey) - (grey_max-grey));
    const int defaultPower; // reference engine power
    int e, u, leftMotor, rightMotor;

    while (true) {
        e = SensorValue[S1] - grey; // momentary error
        u = Kp * e; // regulator
        leftMotor = defaultPower + u; // current power of the left engine
        rightMotor = defaultPower - u; // current power of the right engine
        // control of motors
        if( leftMotor > 100) leftMotor = 100;
        else if( leftMotor < -100) leftMotor = -100;
        if( rightMotor > 100) rightMotor = 100;
        else if( rightMotor < -100) rightMotor = -100;
        // control of motors
        motor[motorB]=leftMotor;
        motor[motorC]=rightMotor;
        wait1Msec(1);
    }
}
```

Движението по линията на П-регулатора се отличава с гладкост, а в някои области роботът се движи почти направо или точно повтаря извивките на линията. При достатъчно голям коефициент на усилване, движението на робота наподобява движението



с релеен регулатор. При ниска стойност на коефициента, роботът може да загуби линията. За да се постигне максимален ефект от регулирането, е необходимо да се изберат оптималните стойности и съотношение между коефициента на усилване  $K_p$  и базова мощност на двигателя.

Вече сме запознати с пропорционалния компонент описан по-горе. Нека да разгледаме интегралната съставяща. Този елемент представлява сума от предишните стойности на грешката:

$$I = I + K_I e(t) dt$$

Физическият смисъл на  $e(t)dt$  е, че той е пропорционален на продължителността на периода, в който системата се намира в състояние на грешка. Тук  $dt$  е времето между две измервания на сензора. Тъй като коефициентът  $K_I$  е поставен извън скобите, можем да говорим за стойността на  $I$  като сума от продължителността на грешките. По този начин намираме интеграла чрез сумиране. Ако грешката поддържа един и същи знак за няколко такта, интегралът нараства. Например, ако проверим светлинния сензор и изчислим, че грешката е 1, то след кратко време отново проверяваме сензора и грешката е 2, след това следващия път, когато грешката отново е 2, тогава интегралът ще бъде  $1 + 2 + 2 = 5$ . Интегралът е 5, но грешката в тази конкретна стъпка е само 2. Интегралът може да бъде голям фактор в корекцията, но обикновено отнема известно време, докато интегралът се изгради до точката, в която започва да допринася за управлението.

Друго нещо, което интегралът прави е, че помага за премахване на малки грешки. Ако в робота следящият линия светлинен сензор е доста близо до края на линията, но не точно върху нея, тогава грешката ще бъде малка и ще се генерира малка корекция. Може да успеем да поправим тази малка грешка като променим  $K_p$  в пропорционалния регулатор, но това често води до колебания в движението на робота. Интегралният регулатор е идеален за фиксиране на малки грешки. Тъй като интегралът добавя към грешката стойностите на няколко последователни малки грешки, което в крайна сметка правят интеграла достатъчно голям, за да направи разлика.

При появата на поредица от еднакви по знак грешки, стойността на интегралната съставяща ще нарасне много. За да се коригира този проблем има две общи решения :

1. Нулиране на интегриращата съставяща всеки път, когато грешката е нула или грешката променя знака си.
2. "Потискане" на интеграла чрез умножаване на натрупания интеграл с коефициент  $(k_I)$ , по-малък от единица  $0 < k_I \leq 1$ , при всяко изчисляване на нов интеграл.

$$I = k_I \cdot I + K_I \cdot e(t) \cdot dt$$

Време е към контролера да добавим и "Δ" съставяща. До момента контролерът съдържа пропорционален ( $P$ ) елемент, който се опитва да коригира текущата грешка и интегрален ( $I$ ) елемент, който се опитва да коригира грешките от миналото.

Има ли начин контролерът да погледне напред във времето и да се опита да поправи грешката, която още не се е случила? На практика ние можем да погледнем в бъдещето, като приемем, че следващата промяна в грешката е същата като последната промяна. Това означава, че следващата грешка се очаква да бъде текущата грешка плюс промяната в грешката между двете предходни измервания на сензора. Промяната в грешката между две последователни измервания се нарича диференциал. Наклонът на линия е също диференциал. Това може да звучи малко сложно за изчисляване, но е доста просто. Да



предположим, че текущата грешка е 1 и грешката преди това е била 3. Какво ще предвидим за следващата грешка? Промяната в грешката за изминалния период е:

$$de(t) = e(t) - e(t - 1)$$

където  $e(t)$  е текущата грешка,  $e(t - 1)$  е предишната грешка, а  $de(t)$  е промяната на грешката за изминалния период, наречен диференциал или първа производна.

За нашия пример получаваме  $de(t) = 1 - 3 = -2$ . Следователно текущата стойност на диференциала е -2. За да определим следващата грешка използваме зависимостта:

$$e(t + 1) = e(t) + de(t) = 1 + (-2) = -1$$

където  $e(t + 1)$  е следващата грешка.

Така, че ние предвиждаме, че следващата грешка ще бъде -1. На практика ние всъщност не вървим по целия път и не предсказваме следващата грешка. Вместо това ние просто използваме производната директно в уравнението на контролера. Физически диференциалът представлява промяната на грешката, разделена на времето, за което е станала тази промяна. Така в нашия алгоритъм трябва да включим и това време  $dt$  наречено такт на „дискретизация“. Точно както с пропорционалните и интегралните съставящи и диференциалната съставяща трябва да се умножава с константа. Тъй като това е константата, която върви с диференцирането, тя се отбележва с  $K_D$ . Забележете също, че диференциалната съставяща разделяме на такта на дискетизация  $dt$ , а интегралната съставяща умножаваме по  $dt$ .

Сега можем да напишем пълното уравнение за ПИД регулатора:

$$u(t) = K_p \cdot e(t) + I(t - 1) + K_I e(t) dt + K_D \frac{e(t) - e(t - 1)}{dt}$$

където  $I(t - 1)$  е стойността на интеграла в предходния момент.

Ако текущата грешка е по-лоша от предишната, то диференциалната съставяща се опитва да коригира грешката. Ако текущата грешка е по-добра от предходната грешка, тогава диференциалната съставяща се опитва да спре контролера да коригира грешката. Ако грешката се доближава до нула, тогава ние се приближаваме до точката, в която искаме да спрем коригирането. Тъй като системата вероятно отнема известно време, за да реагира на промените в мощността на двигателите, ние искаме да започнем да намаляваме мощността на двигателя, преди грешката действително да е достигнала нула. В противен случай ще подминем нулата.

Ето и кода, реализиращ ПИД закон за управление на робот следящ линия:

```

task main(){
    const float grey_min = 43; // minimum value of gray
    const float grey_max = 47; // maximum value of gray
    const int grey = (grey_max + grey_min)/2; // the value of gray
    const float Kp = (1 - (-1)) / ((grey_min-grey) - (grey_max-grey));
    const float Ki = 1; // integration coefficient
    const float Kd = 1; // differential coefficient
    const int defaultPower; // default engine power
    int e,errold=0, u, leftMotor, rightMotor;
    float I = 0, dT= 0.001;
    while (true) {
        e = SensorValue[S1] - grey; // current error
        P = Kp * e; // proportional component
        // Reseting of integral component
        if( e == 0 || (e/abs(e)) != (errold /abs(errold)) ) I = 0;
        I = I + Ki*e* dT; // integral component
        D = Kd*(e - errold)/ dT; // differential component
        u = P + D + I;
        leftMotor = defaultPower + u; //current power of the left motor
        rightMotor= defaultPower - u; //current power of the right motor
        // control of motors
        if( leftMotor > 100) leftMotor = 100;
        else if( leftMotor < -100) leftMotor = -100;
        if( rightMotor > 100) rightMotor = 100;
        else if( rightMotor < -100) rightMotor = -100;
        // control of motors
        motor[motorB]=leftMotor;
        motor[motorC]=rightMotor;
        wait1Msec(1);
    }
}

```

Вече имаме кода на ПИД контролер за следене на линия. Сега идва по-трудната част, “настройване” на ПИД контролера. Настройката е процесът на намиране на най-добрите или поне нормални стойности за  $K_p$ ,  $K_i$  и  $K_d$ .

За да настроите PID контролера, изпълнете следните стъпки, базирани на метода на Ziegler–Nichols за настройка на регулатори:

1. Задайте на  $K_i$  и  $K_d$  стойност нула, което изключва тези съставящи и прави регулатора да действа като обикновен П регулатор.
2. Задайте малка стойност на базова мощност на двигателя defaultPower например 25.
3. Задайте на пропорционалната съставяща  $K_p$  “разумна” стойност. Първоначалната стойност на  $K_p$  можем да подберем по следните начини:
  - взема се максималната стойност, която искаме да зададем на драйвера на двигателя (100) и я разделяме на максималната използваема грешка. За нашия робот за следене на линия сме приели, че максималната грешка е 5, така че предполагаме, че  $K_p = 100/5 = 20$ . Когато грешката е +5, мощността на двигателя ще се върти с 100 единици. Когато грешката е нула, мощността на двигателя ще се намира на нивото на базовата мощност defaultPower.
  - или задайте на  $K_p$  стойност 1 (според метода на Ziegler–Nichols) или 100 и вижте какво се случва.
4. Пуснете робота и наблюдавайте какво прави. Ако не може да следва линията и се отклонява се увеличава  $K_p$ . Ако много осцилира, тогава намалете  $K_p$ . Продължавайте да променяте стойността на  $K_p$ , докато намерите такава, при която следва линията и



дава забележими колебания, но не и резки. Ще наричаме тази стойност  $K_{kp}$  ("критично усилване").

5. Задайте  $K_{kp}$  като стойност на  $K_p$ , стартирайте робота по линията и се опитайте да определите колко бързо се колебае. Периодът на колебанията ( $T_{kp}$ ) е времето, за което роботът се залюлява от крайно ляво до крайно дясна позиция и обратно. За Lego роботите ТкР приема стойност в диапазона от около 0,5 секунди до 1-2 секунди.
6. На следващата стъпка се определя тактът на дискретизация  $dT$ . За  $dT$  може да се засече времето за изпълнение на определен брой работни цикли например 10 000 на броя итерации.
7. Използвайте таблицата по-долу, за да изчислите стойностите на  $K_p$ ,  $K_I$  и  $K_D$ . Ако просто искате П-контролер, използвайте реда от таблицата с П, за да определите  $K_p$  ( $K_I = 0$ , и  $K_D = 0$ ). Ако искате ПИ-контролер, използвайте следващия ред и т.н.
8. Задайте изчислените стойности на коефициентите на регулатора. Стартирайте робота и вижте как се държи.
9. Настройте  $K_p$ ,  $K_I$  и  $K_D$  стойностите така, че да получите най-доброто движение на робота. Стойността на коефициентите се коригират една по една като се започва от  $K_p$ . Стойностите се променят докато спре подобряването поведението на робота. След това се настройват последователно  $K_I$  и  $K_D$ .
10. След като имате добър набор от коефициенти на регулатора, се опитайте да увеличите стойността на базовата мощност на двигателя defaultPower, която контролира правилната скорост на робота.
11. Донастройте коефициентите на регулатора или може би дори се върнете към стъпка 1 и повторете целия процес за новата стойност на defaultPower.
12. Продължавайте да повтаряте, докато поведението на робота не стане приемливо.

*Таблица 3: Коефициенти за настройка на ПИД регулатор по метода на Ziegler–Nichols*

Вид регулатор	$K_p$	$K_I$	$K_D$
П	$0.50K_{kp}$	0	0
ПИ	$0.45 K_{kp}$	$1.2K_p dT / T_{kp}$	0
ПД	$0.80 K_{kp}$	0	$K_p T_{kp} / (8dT)$
ПИД	$0.60 K_{kp}$	$2K_p dT / T_{kp}$	$K_p T_{kp} / (8dT)$

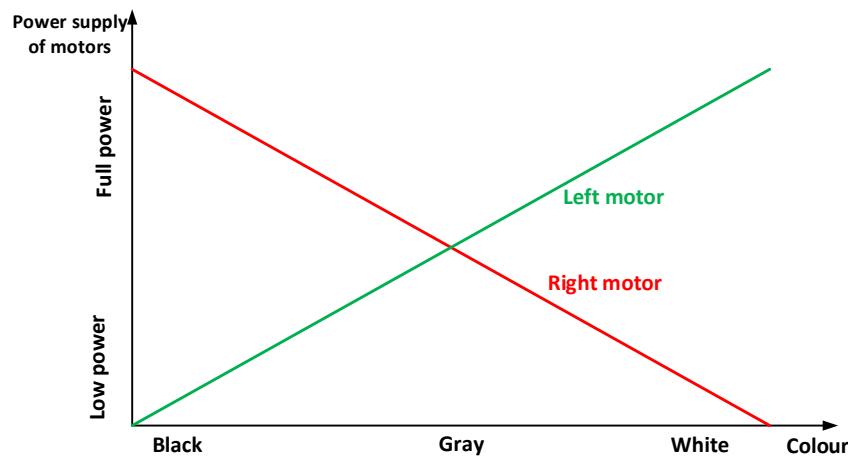
#### • Размито управление на робот за следене на линия

Демонстрираните по-горе алгоритми за следене на линия използват отразената светлина, за да се различи черно (линия) и бяло (фон). Всеки път, когато сензорът за осветление е над черната линия, левият двигател се активира и кара робота да завие на дясно. Когато сензорът е над бялата повърхност, десният мотор се активира и роботът завива наляво. По този начин роботът следва линията или за да бъдем по-точни, той следва границата между черно и бяло. Подходът е много прост и ефективен. Въпреки това, тъй като роботът различава само черно и бяло, резултатът от движението му изглежда така, сякаш той е пиян. Идеята за новата версия на управлението е не само да се прави разлика между черно и бяло, а да се разглеждат различни степени на сиво. Ако сензорът е точно на границата, той ще види цвят, който е между черно и бяло. Колкото повече сензорът попада върху



линията, толкова по-черен е възприеманият цвят. От друга страна, колкото повече сензорът попада върху бялата повърхност, толкова по-бял е възприеманият цвят. Това се нарича размита логика (Fuzzy logic), тъй като не се разглеждат абсолютните състояния ("черно и бяло"), а различни степени на тези състояния (по-бели, по-черни). По този начин точно по границата на цветовете роботът ще върви право напред. За да се постигне това, левият и десният двигатели се управляват съгласно диаграмата, показана на фигурана по-долу.

Сензорът на светлина връща стойности между 0 и 100. Обикновено стойността, върната при черна повърхност е около 20, а за бяла повърхност - около 60. Тези стойности зависят от условията на околната среда и от разстоянието на сензора до повърхността. Самата мощност на двигателя също има стойност между 0 и 100.



За да се изведат командите на двигателите, както е описано в схемата, трябва да се направят някои изчисления.

1. Да се определят стойностите, връщани от сензора за светлина, отговарящи на черно (Black\_Value) и бяло (White\_Value).
2. Да се определи максималната (maxSpeed) и минималната (minSpeed) стойност на сигнала, подаван към двигателите.
3. Необходимо е да определим коефициента на скалиране Speed\_Factor по формулата.

$$\text{Speed\_Factor} = \frac{\text{maxSpeed} - \text{minSpeed}}{\text{White\_Value} - \text{Black\_Value}} = \frac{100 - 0}{60 - 20} = \frac{100}{40} = 2.5$$

4. Да се определят моментните стойности на сигналите, подавани към двигателите, където скоростта на движение на колелата се задава съответно от Left\_Power и Right\_Power

$$\text{Left\_Power} = (\text{Sensor\_Value} - \text{Black\_Value}) * \text{Speed\_Factor}$$

$$\text{Right\_Power} = (\text{White\_Value} - \text{Sensor\_Value}) * \text{Speed\_Factor}$$

Кодът на програмата, реализиращ метода, е следният:



```

task main(){
    float Black_Value = 20;
    float White_Value = 60;
    float maxSpeed = 100;
    float minSpeed = 0;
    float Speed_Factor=(maxSpeed-minSpeed)/(White_Value-Black_Value);
    int Left_Power,Right_Power, Sensor_Value;
    while (true) {
        Sensor_Value = SensorValue[S1] ;
        Left_Power = (Sensor_Value-Black_Value)*Speed_Factor;
        Right_Power = (White_Value-Sensor_Value)*Speed_Factor;
        // control of motors
        if( Left_Power > 100) Left_Power = 100;
        else if( Left_Power < -100) Left_Power = -100;
        if( Right_Power > 100) Right_Power = 100;
        else if( Right_Power < -100) Right_Power = -100;
        // control of motors
        motor[motorB]=Left_Power;
        motor[motorC]=Right_Power;
        wait1Msec(1);
    }
}

```

Ако програмирате вашия робот с тази програма и роботът не работи както се очаква, проверете следното:

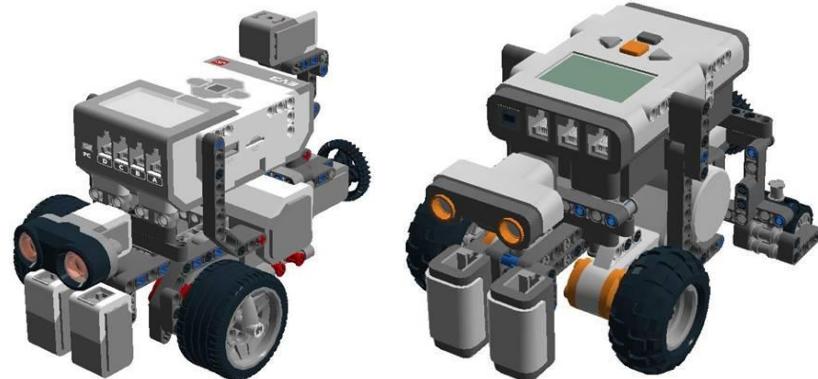
1. Позиционирайте светлинния сензор на разстояние 1cm от повърхността.
2. Прочетете стойностите, връщани от светлинния сензор при черна и бяла повърхност и ги заложете в програмата.
3. Задайте минималната и максимална стойност подавани към драйвера, управляващ двигателите на робота.
4. За фина настройка на скоростта на движение започнете да променяте стойността на коефициента за скалиране Speed\_Factor.

#### • Алгоритъм за следене на линия с два сензора

До момента разгледаните алгоритми са за роботи с един сензор. Водещият сигнал е разликата между черната линия и белия фон. Целта е да се поддържа „сиво“ ниво, което да показва, че сензорът е на края на линията, следвайки или десния или левия край, докато роботът се движи напред. Какво би станало, ако към робота добавим още един светлинен сензор и как трябва да бъдат разположени сензорите?

За да отговорим на този въпрос е необходимо да си отговорим на въпроса „Кой цвят ще следим, цвета на линията или цвета на фона? В зависимост от отговора се определя и конструкцията на робота. Да предположим, че желаем да следим цвета на фона. При това положение сензорите трябва да се разположени от двете страни на линията.

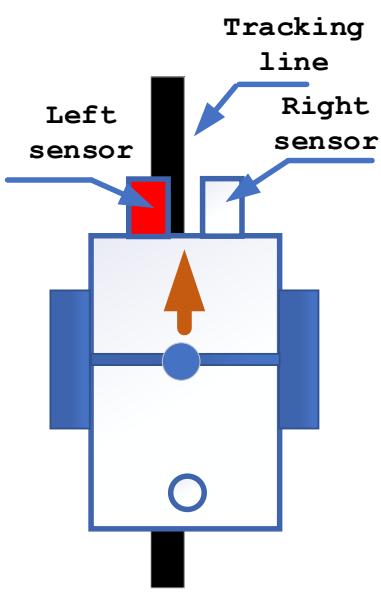
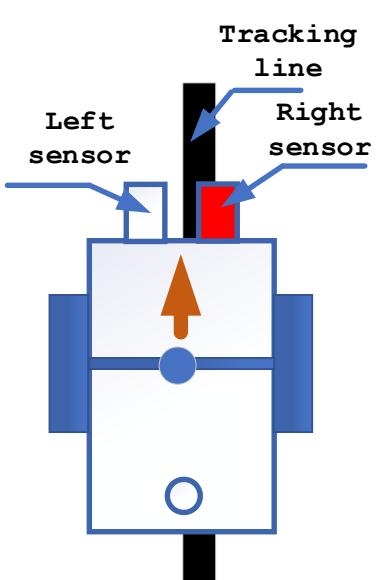




Конструкция на робот за следене на линия с два сензора

Роботът, следващ линията, е програмиран така, че основно да се избегне черният цвят. При тази ситуация сензорите се поставят от двете страни на линията за проследяване. Ако и двата сензора виждат черно, роботът ще се върне, докато вече не вижда черно. Ако един сензор вижда черно, роботът ще се обърне по посока на този сензор, докато вече не вижда черно. Освен това, ако роботът не вижда черно на нито един от сензорите, той просто ще се движи напред. Логиката за този алгоритъм е показана по-долу заедно с графично представяне на робота. Има четири възможни състояния на сензорите, при които роботът реагира.

	<p><b>Случай 1:</b></p> <p>В този случай и двета сензора не откриват линията. И двата двигателя се въртят напред. В резултат на това роботът се движи напред.</p> <pre> LeftSensorValue = SensorValue[S1] ; RightSensorValue = SensorValue[S2] ; if( LeftSensorValue &gt; Black_Value &amp;&amp;     RightSensorValue &gt; Black_Value ) {     // move forward     Left_Power = maxSpeed;     Right_Power = maxSpeed; } </pre>
--	--

	<p><b>Случай 2:</b></p> <p>В този случай само левият сензор открива линията, което означава, че роботът трябва да се завърти в лявата посока. Левият двигател се върти назад и десният мотор се върти напред. В резултат на това роботът завива наляво.</p> <pre> if( LeftSensorValue &lt;= Black_Value &amp;&amp;     RightSensorValue &gt; Black_Value ) {     // move left     Left_Power = -20;     Right_Power = 80; } </pre>
	<p><b>Случай 3:</b></p> <p>В този случай само десният сензор открива линията, което означава, че роботът трябва да се завърти в правилната посока. Левият двигател се върти напред и десният се върти назад. В резултат на това роботът завива надясно.</p> <pre> if( LeftSensorValue &gt; Black_Value &amp;&amp;     RightSensorValue &lt;= Black_Value ) {     // move right     Left_Power = 80;     Right_Power = -20; } </pre>

	<p><b>Случай 4:</b></p> <p>В този случай и двата сензора откриват линията. Това означава, че е дошъл краят. И двата двигателя спират да се въртят. В резултат на това роботът спира или завива наляво в зависимост от изискванията на задачата .</p> <pre> if( LeftSensorValue &lt;= Black_Value &amp;&amp;     RightSensorValue &lt;= Black_Value ) {     // stop movements     Left_Power = 0;     Right_Power = 0; } </pre>
--	--

Ето и кода на цялата програма:



```

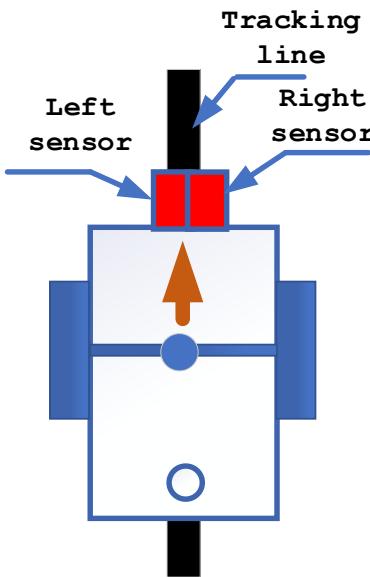
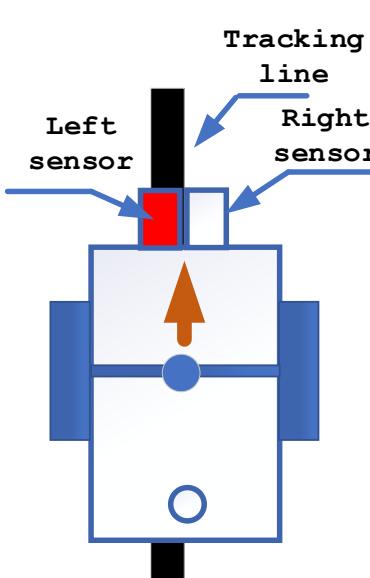
task main(){
    float Black_Value = 20;
    float White_Value = 60;
    float maxSpeed = 100;
    float minSpeed = 0;
    int Left_Power,Right_Power, LeftSensorValue,RightSensorValue;
    while (true) {
        LeftSensorValue = SensorValue[S1] ;
        RightSensorValue = SensorValue[S2] ;
        if( LeftSensorValue > Black_Value &&
            RightSensorValue > Black_Value ) {
            // move forward
            Left_Power = maxSpeed;
            Right_Power = maxSpeed;
        } else if( LeftSensorValue <= Black_Value &&
            RightSensorValue > Black_Value ) {
            // move left
            Left_Power = -20;
            Right_Power = 80;
        } else if( LeftSensorValue > Black_Value &&
            RightSensorValue <= Black_Value ) {
            // move right
            Left_Power = 80;
            Right_Power = -20;
        } else if( LeftSensorValue <= Black_Value &&
            RightSensorValue <= Black_Value ){
            // stop movements
            Left_Power = 0;
            Right_Power = 0;
        }
        // control of motors
        if( Left_Power > 100) Left_Power = 100;
        else if( Left_Power < -100) Left_Power = -100;
        if( Right_Power > 100) Right_Power = 100;
        else if( Right_Power < -100) Right_Power = -100;
        // control of motors
        motor[motorB]=Left_Power;
        motor[motorC]=Right_Power;
        wait1Msec(1);
    }
}

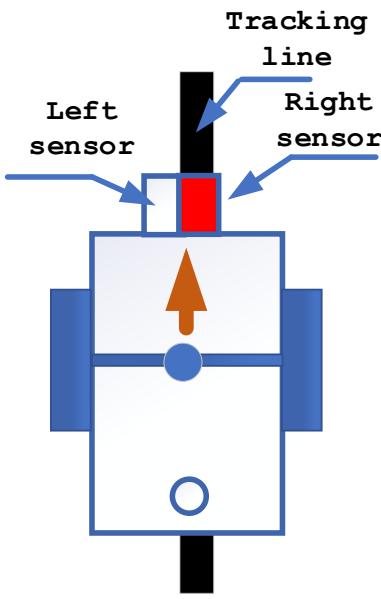
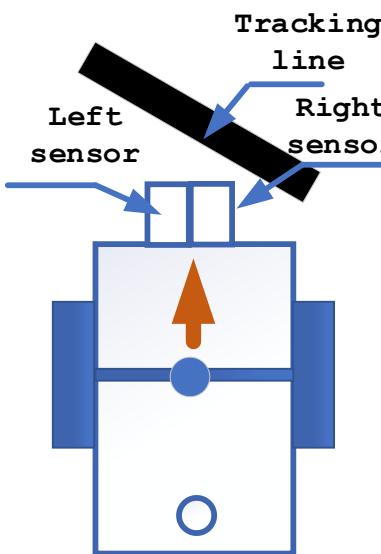
```

В следващия пример роботът, следващ линията, е програмиран така, че основно да се избегне белия цвят. Сензорите се монтират един до друг над линията за проследяване. Ако и двата сензора виждат бяло, роботът ще се върне, докато вече не вижда бяло. Ако един сензор вижда черно, роботът ще се обърне по посока на този сензор, докато двата сензора започнат да виждат черно. Ако и двата сензора виждат черно, роботът ще се движи направо, докато вижда черно.

Както при предния случай и тук има четири възможни състояния на сензорите, при които работят реагира.



 <p>Left sensor      Tracking line      Right sensor</p>	<p><b>Случай 1:</b></p> <p>В този случай и двата сензора откриват линията. И двата двигателя се въртят напред. В резултат на това роботът се движи напред.</p> <pre>LeftSensorValue = SensorValue[S1] ; RightSensorValue = SensorValue[S2] ; if( LeftSensorValue &lt;= Black_Value &amp;&amp;     RightSensorValue &lt;= Black_Value ) {     // move forward     Left_Power = maxSpeed;     Right_Power = maxSpeed; }</pre>
 <p>Left sensor      Tracking line      Right sensor</p>	<p><b>Случай 2:</b></p> <p>В този случай само левият сензор открива линията, което означава, че роботът трябва да се завърти в лявата посока. Левият двигател се върти назад и десният се върти напред. В резултат на това роботът завива наляво.</p> <pre>if( LeftSensorValue &lt;= Black_Value &amp;&amp;     RightSensorValue &gt; Black_Value ) {     // move left     Left_Power = -20;     Right_Power = 80;</pre>

	<p><b>Случай 3:</b></p> <p>В този случай само десният сензор открива линията, което означава, че роботът трябва да се завърти в правилната посока. Левият двигател се върти напред и десният се върти назад. В резултат на това роботът завива надясно.</p> <pre> if( LeftSensorValue &gt; Black_Value &amp;&amp;     RightSensorValue &lt;= Black_Value ) {     // move right     Left_Power = 80;     Right_Power = -20; } </pre>
	<p><b>Случай 4:</b></p> <p>В този случай и двата сензора не откриват линията. Това означава, че роботът е излязъл от линията. Стратегията за търсене е да започне завой наляво докато не открие черна линия.</p> <pre> if( LeftSensorValue &gt; Black_Value &amp;&amp;     RightSensorValue &gt; Black_Value ) {     // search the line     Left_Power = -20;     Right_Power = 60; } </pre>

Ето и кода на цялата програма:

```

task main(){
    float Black_Value = 20;
    float White_Value = 60;
    float maxSpeed = 100;
    float minSpeed = 0;
    int Left_Power,Right_Power, LeftSensorValue,RightSensorValue;
    while (true) {
        LeftSensorValue = SensorValue[S1];
        RightSensorValue = SensorValue[S2];
        if( LeftSensorValue <= Black_Value &&
            RightSensorValue <= Black_Value ) {
            // move forward
            Left_Power = maxSpeed;
            Right_Power = maxSpeed;
        } else if( LeftSensorValue <= Black_Value &&
            RightSensorValue > Black_Value ) {
            // move left
            Left_Power = -20;
            Right_Power = 80;
        } else if( LeftSensorValue > Black_Value &&
            RightSensorValue <= Black_Value ) {
            // move right
            Left_Power = 80;
            Right_Power = -20;
        } else if( LeftSensorValue > Black_Value &&
            RightSensorValue > Black_Value ) {
            // search the line
            Left_Power = -20;
            Right_Power = 60;
        }
        // control of motors
        if( Left_Power > 100) Left_Power = 100;
        else if( Left_Power < -100) Left_Power = -100;
        if( Right_Power > 100) Right_Power = 100;
        else if( Right_Power < -100) Right_Power = -100;
        // control of motors
        motor[motorB]=Left_Power;
        motor[motorC]=Right_Power;
        wait1Msec(1);
    }
}

```

#### ВЪПРОСИ ЗА САМОПРОВЕРКА

1. При следене на линия се използва:
  - a) Сензор за допир
  - б) Сензор за цвят
  - в) Инфрачервен сензор
  - г) Сензор за откриване на обекти
  - д) Сензор за светлина

2. В алгоритъма за следене на линия с един сензор се следи:
  - а) левият ръб на линията
  - б) десният ръб на линията
  - в) левият или десния ръб на линията



г) следи се цвета на линията

3. Кой алгоритъм за следене на линия се нарича „класически”?

- а) релеен регулатор за движение по линия при две зони
- б) релеен регулатор за движение по линия при три зони
- в) ПИД регулатор за движение по линия
- г) Размит регулатор за движение по линия

4. В оптимизирания основен алгоритъм за следване на линията се използва:

- а) релеен регулатор за движение по линия при две зони
- б) релеен регулатор за движение по линия при три зони
- в) ПИД регулатор за движение по линия
- г) Размит регулатор за движение по линия

5. Производната на грешката се използва при:

- а) релеен регулатор за движение по линия при две зони
- б) релеен регулатор за движение по линия при три зони
- в) ПИД регулатор за движение по линия
- г) Размит регулатор за движение по линия

6. Термините "по-бяло", "по-малко бяло" се използват при реализирането на:

- а) релеен регулатор за движение по линия при две зони
- б) релеен регулатор за движение по линия при три зони
- в) ПИД регулатор за движение по линия
- г) Размит регулатор за движение по линия

7. В алгоритъма за следене на линия с два сензора се следи:

- а) левият ръб на линията
- б) десният ръб на линията
- в) левият или десния ръб на линията
- г) левият и десният ръб на линията

*Верни отговори: 1 (д); 2 (б); 3 (а); 4 (б); 5 (б); 6 (с); 7 (г)*

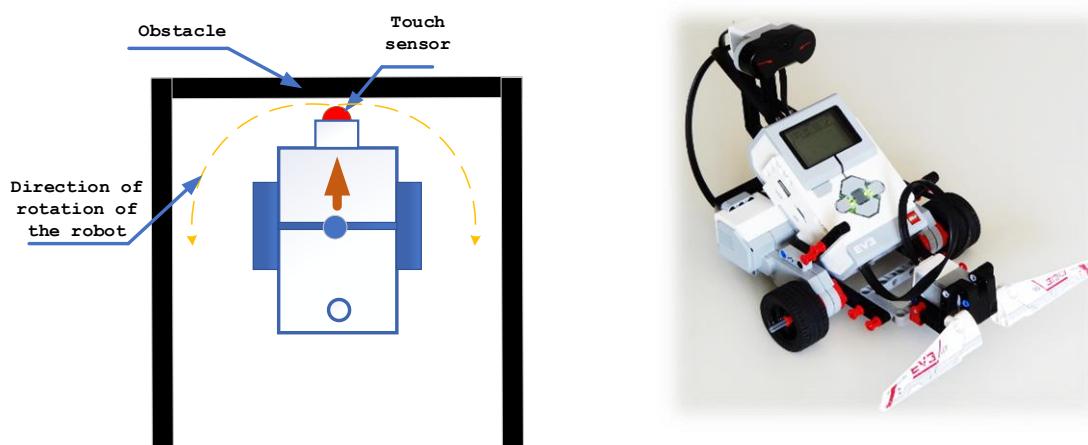


## Избягване на препятствия

За избягване на препятствия съществуват множество стратегии. Тук ще се спрем на три от тях. При първата стратегия на робота се монтира сензор за допир. При задействане на сензора, роботът променя траекторията си на движение. При втората стратегия, роботът е снабден със сензор, определящ разстоянието до обектите. При тази стратегия не се стига до пряк сблъсък с обекта, а той се заобикаля. Третата стратегия е комбинация от първите две. При нея сензорът за допир се използва за избягване на сблъсък, ако сензорът за разстояние пропусне някой обект.

### ИЗБЯГВАНЕ НА ПРЕПЯТСТВИЯ СЪС СЕНЗОР ЗА ДОПИР

С помощта на сензор за допир роботът може да открива **прегради пред себе си** и да предотвратява сблъсък с тях. За целта ще използваме двуколесен робот с трето помошно колело (виж фигурата по-долу). В предната част на робота се монтира сензор за допир. Сензорът за допир е цифров сензор, който реагира при натиск върху чувствителния му елемент. Това означава, че сензорът за допир може да бъде програмиран да действа в зависимост от три условия: натискате, освобождаване и кликване (натискане и освобождаване). Благодарение на информацията, получена от сензора за допир, може да се програмира роботът така, че да вижда какво го заобикаля по същият начин както го правят слепите хора: опознават околното пространството, като реагират на допир (натиск) до предметите.



Избягване на препятствие от двуколесен робот с един сензор за допир

Също така може да се изгради робот със сензор за допир, който е ориентиран надолу и се притиска към повърхността под него. Робота се програмира така, че да спре движението си когато сензорът за допир се освободи. Освобождаването означава, че под нивото на робота има дупка или е достигнат краят на масата, върху която е поставен роботът.

Подходът, който ще предприемем за реализиране на алгоритъма за избягване на препятствия се базира на следните съждения: ако се намерят препятствия по пътя на робота, роботът ще се върне малко назад и ще опита движение в нова посока (наляво или надясно в зависимост от избраната стратегия на обхождане), опитвайки се да избегне препятствието. Това позволява на робота да изследва райони, без да засяде или да се повреди.

За да се получи точно завиване на  $90^\circ$  или друг ъгъл, може да се използват два подхода. При първия единият мотор се движи в права посока, а другият в обратна в продължение на зададено време. Този подход е лесен за реализация и точността на изпълнение зависи силно от заряда на батериите.

При втория подход се използва вграденият енкодер за определяне ъгъла на завъртане. Енкодерът позволява да се измерва ъгълът на завъртане на двигателя с точност от 1 градус. При реализирането на програмата трябва да имате в предвид, че за да се завърти робота на 90 градуса около оста си трябва лявото колело да се завърти на 250 градуса:

```
task main() {
    // turn 90 degrees to the left
    nMotorEncoder[motorB]=0; // initialization of the encoder
    motor[motorB] = 100;
    motor[motorC] = -100;
    // Empty loop waiting for encoder readings
    while(nMotorEncoder[motorB]<250);
    motor[motorB] = 0;
    motor[motorC] = 0;
}
```

Когато завивате наляво или надясно, роботът продължава да проверява за препятствията по пътя си. Ако открие препятствия, той ще спре и ще повтори завъртация цикъл, докато не открие проход.

Последователността на изпълнение е следната:

1. Стартурайте драйвера на робота и изчакайте 5 секунди. Това ви дава време да поставите робота там, където искате, преди да започне да се движи.
2. Роботът се придвижи напред.
3. Ако сензорът се активира, роботът спира и се връща назад за 1 секунда, след което се завърта наляво или надясно според стратегията на обхождане.
4. Ако отново сензорът е активиран, се преминава към стъпка 3.
5. Ако сензорът не е активиран, се преминава към стъпка 2.

Ето и програмната реализация на алгоритъма:



```

task main() {
    while( true ) {
        wait1Msec(5000);
        // move forward
        motor[motorB] = 100;
        motor[motorC] = 100;
        if ( SensorValue[S2] == 1 ) { // check touch sensor
            // stop the motors
            motor[motorB] = motor[motorC] = 0;
            // return back
            motor[motorB] = motor[motorC] = -100;
            wait1Msec(1000);
            // Turn 90 degrees to the left
            nMotorEncoder[motorB]=0; // initialization of the encoder
            motor[motorB] = 100;
            motor[motorC] = -100;
            // Empty loop waiting for encoder readings
            while(nMotorEncoder[motorB]<250);
            motor[motorB] = motor[motorC] = 0;
        }
    }
}

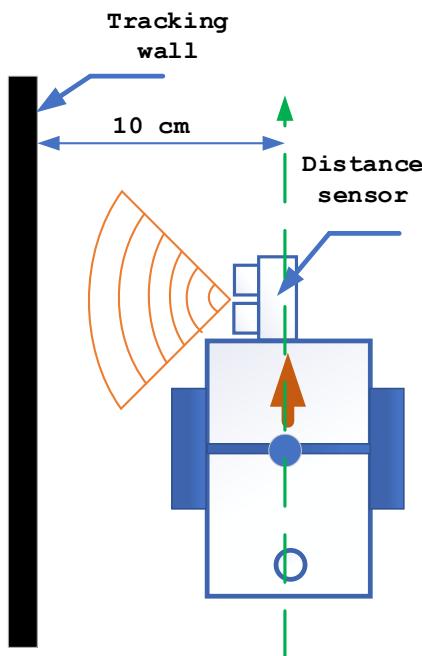
```

### ДВИЖЕНИЕ ПО СТЕНА

Движение по стена с малки отклонения е възможно с помощта на програмата за следене на линия с PD регулатор, разгледана в предишния раздел. Разликата между двете програми е, че в едната се използва сензор за разстояние, а в другата сензор за цвят. Останалата част от програмите са сходни: роботът променя посоката на движение след промяна на стойността на сензора. По-точно, в задачата роботът се обръща надясно, ако разстоянието на сензора е по-малко от 9 см и наляво, ако разстоянието на сензора е повече от 11 см (виж фигурана по-долу).

Ако начертаете диаграма на такова движение, става ясно, че от лявата страна на робота по време на цялото движение има преграда и роботът се опитва да не се движи твърде близо до нея или твърде далеч от нея. Ако приемем, че препятствието е стена, то движението на робота може да се нарече движение по стената. В случай на неголямо изкривяване на стената, роботът ще се опита да се задържи на зададеното разстояние.





*Движение на робот по стена*

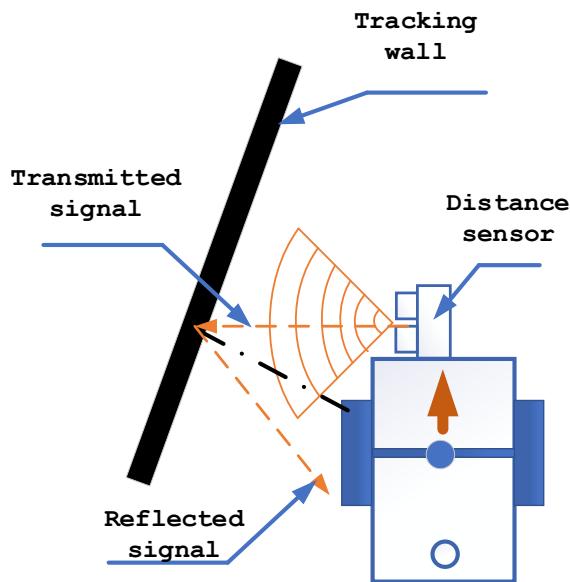
Алгоритмите за следене на линия и следене на стена са базови алгоритми при обучението по роботика. Т.е. при решаването на реални проблеми алгоритъмът на движение ще бъде много по-сложен, но принципът на движение ще остане същият.

Тук ще покажем някои случаи, които често са „препъни камъни“ за онези, които току що започват да реализират програми за движение по стена.

Като цяло, движението на робота успоредно на стената се базира на показанията на сензора за разстояние. Основната цел на ултразвуковия сензор е да определи разстоянието до обектите пред него. За да направите това, сензорът изпраща високочестотна звукова вълна (ултразвук), улавя отразената от обекта вълна и измервайки времето за връщане на ултразвуковия сигнал, изчислява разстоянието до обекта.

Ултразвуковият сензор може да изведе измереното разстояние в сантиметри или инчове. Диапазонът на измерване на сензора в сантиметри е от 0 до 255 см, в инчове - от 0 до 100 инча. Сензорът не може да открива обекти на разстояние по-малко от 3 см (1,5 инча). Също така, той не измерва достатъчно точно разстоянието до порести прегради, тъкани и обекти с малък обем. В допълнение към режимите за измерване на разстоянието в сантиметри и инчове, сензорът има специален режим "Присъствие/Слушане". В този режим сензорът не излъчва ултразвукови импулси, но може да открие импулсите на друг ултразвуков сензор.

В някои случаи е възможно информацията постъпваща от сензора да е неточна. Например, ако роботът не е разположен успоредно на преградата (виж фигурата по-долу), излъченият сигнал от сензора ще се отрази от повърхността под ъгъл не позволяващ на отразената звукова вълна да попадне върху сензора и той ще „мисли“, че препятствието е все още твърде далече. Това ще накара сензора да покаже много голямо разстояние до обекта.



*Грешка при отчитане на разстоянието при движение на робот по стена*

В този случай, роботът ще се опита да се доближи до стената, увеличавайки ъгъла между сензора и стената, което само ще влоши непцата.

Съществуват много решения на този проблем. Те могат да бъдат както програмни, така и конструктивни. Например, вие можете да не фиксирате сензора твърдо, а да го поставите на двигателя.

Така, след завъртане на робота, например наляво, сензорът за разстояние се върти, опитвайки се да бъде насочен директно към стената. И когато завивате надясно, двигателят завърта сензора в другата посока.

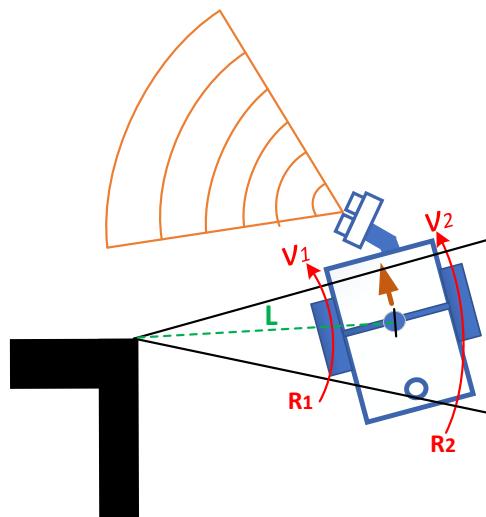
Тази схема е удобна при сглобяване на робот с мотор за кормилно управление, след което сензорът може да се монтира към същия двигател, който контролира водещите колела. Освен това е по-добре да не се монтира директно, а да се избере подходяща комбинация от предавки.

Описаният робот обаче може да обикаля стени само с малки отклонения от правата линия. При остри завои роботът може да загуби контакт със стената и да започне да се върти на място. Този проблем може да бъде частично решен конструктивно, като сензорът за разстояние се монтира под остър ъгъл спрямо посоката на движение на робота или софтуерно.

**Важна забележка.** Когато роботът се стартира, той трябва да бъде насочен успоредно на стената, така че отчитането на първоначалната стойност да преминава без интерференция.

Появата на завой ще доведе до рязка промяна на разстояние до обекта. По това роботът разбира, че е възможно наличието на завой. За да се продължи следенето на линията, движението ще трябва да се направи с други коефициенти или просто чрез управление с пропорционален закон. Да разгледаме примера за завиване на ляво (виж фигурата подолу).

Ако роботът се движи на разстояние  $L$  от стената, тогава той очевидно ще се обърне по окръжност с радиус  $L$ . Лесно е да се изчисли каква трябва да бъде скоростта на колелата за да се завърти роботът с радиус  $L$  около ъгъла на стената.



Заобикаляне около ъгъл на стената

Достатъчно е да се измери разстоянието между предните колела. Нека тя е равна на  $k = 12 \text{ cm}$  за нашия робот, а половината му  $d = 12 / 2 = 6 \text{ cm}$ , след което лявото и дясното колело се движат по окръжности с радиуси, съответно  $R1 = L - d$  и  $R2 = L + d$ . Пътищата, изминати от тях за единица време, трябва да бъдат пропорционални на радиусите, следователно, скоростта на точките на закрепване на колелата е  $v_1$  и  $v_2$ ,  $\frac{v_1}{v_2} = \frac{R_1}{R_2}$ . Изразявайки скоростта на движение на колелата през основната скорост  $v$  и неизвестната  $x$ , и радиусите през  $L$ , получаваме следното:

$$\frac{v+x}{v-x} = \frac{L+d}{L-d}, \quad vL + xL - vd - xd = vl + vd - xL - xd, 2xL = 2vd,$$

$$x = \frac{vd}{L}, \quad v_1 = v - \frac{vd}{L} = v\left(1 - \frac{d}{L}\right), \quad v_2 = v + \frac{vd}{L} = v\left(1 + \frac{d}{L}\right)$$

Линейната скорост  $v$  е пропорционална на ъгловата скорост на колелото  $\omega$ , която от своя страна е пропорционално на мощността в режим на натоварване, която се подава към двигателите. Тук представихме закона за управление в стандартна форма, който позволява да бъде синтезирано управляващо въздействие в реално време.

```
u=v*6/L;
motor[motorB]=v-u;// left motor
motor[motorC]=v+u;// right motor
```

Когато разстоянието до стената стане по-голямо от  $2L$ , т.е. появил се е завой, управляващото въздействие се определя по дадените формули.

Но в някой ситуации е възможно да се получат резки промени в показанията на сензора. Това би довело до рязка промяна в поведението на робота. Тогава филтрирането на данните става особено важно за по-нататъшни действия в дългосрочен план. Наличието на един филтър би изчистило тези флуктуации във входния сигнал. Ето зато, филтрите, въпреки че забавят реакцията на робота, но за сметка на това го правят по-стабилен и предвидим. При решаване на задачата по следене на стена е достатъчно да бъде използван нискочестотен цифров филтър описан с рекурентното уравнение:

$$y(n) = (1 - a) y(n - 1) + a \cdot x(n)$$



В примерната програма показана по-долу, цифровият филтър е реализиран със следният код.

```
Snew=(1-a)*Snew+a*SensorValue[S1];
```

По този начин нашата програма става по-устойчива на смущения.

```
task main()
    float u, k1=2, k2=10, a=0.2;
    int d = 8; // Half the width of the robot
    int defaultPower = 50; // default power of the motor
    int Sold, L, Snew, leftMotor, rightMotor;
    wait1Msec(5000); //waiting installation of robot on position
    Sold = L = SensorValue[S1]; // remember the initial position
    while(true) {
        // receive and filter sensor readings
        Snew=(1-a)*Snew+a*SensorValue[S1];
        if (Snew>L*2) {
            // turning motion
            u=defaultPower*d/L;
            Sold=L*2;
        } else {
            // normal motion
            u = k1*(Snew-L) + k2*(Snew-Sold);
            Sold=Snew;
        }
        leftMotor = defaultPower + u;
        rightMotor = defaultPower - u;
        // control of motors
        if( leftMotor > 100) leftMotor = 100;
        else if( leftMotor < -100) leftMotor = -100;
        if( rightMotor > 100) rightMotor = 100;
        else if( rightMotor < -100) rightMotor = -100;
        // control of motors
        motor[motorB]=leftMotor;
        motor[motorC]=rightMotor;
        wait1Msec(1);
    }
}
```

Очевидно е, че промяната в конструкцията води до промяна в коефициентите на регулатора  $k_1$  и  $k_2$ . Обикновено настройката започва с пропорционалния коефициент при нулев диференциал. Когато се постигне известна стабилност при малки отклонения, се добавя диференциална компонента.

### ВЪПРОСИ ЗА САМОПРОВЕРКА

1. При следене на стена се използва:
  - a) Сензор за допир;
  - б) Сензор за цвят;
  - в) Инфрачервен сензор;
  - г) Сензор за откриване на обекти;
  - д) Сензор за разстояние.
2. Кой сензор се използва за откриване на дупка в пода под робота:
  - a) Сензор за допир;



- б) Сензор за цвят;
- в) Инфрачервен сензор;
- г) Сензор за откриване на обекти;
- д) Сензор за разстояние.

3. Кои от изброените алгоритми за следене на линия могат да се приложат при следена на стена?

- а) релеен регулатор за движение по линия при две зони;
- б) релеен регулатор за движение по линия при три зони;
- в) ПИД регулатор за движение по линия;
- г) Размит регулатор за движение по линия.

4. При следене на стена с един сензор, сензора за разстояние се поставя:

- а) право напред по движение ето на робота;
- б) перпендикулярно по движението на робота в посока на стената;
- в) под остръ тъгъл спрямо по движението на робота и стената;
- г) сензорът се поставя на двигател и се върти заедно с робота.

5. При определяне закона за движение на робот за следене на линия се използват тригонометричните функции:

- а) периметър на окръжност;
- б) уравнение на линия в равнината;
- в) уравнение на линия в пространството.

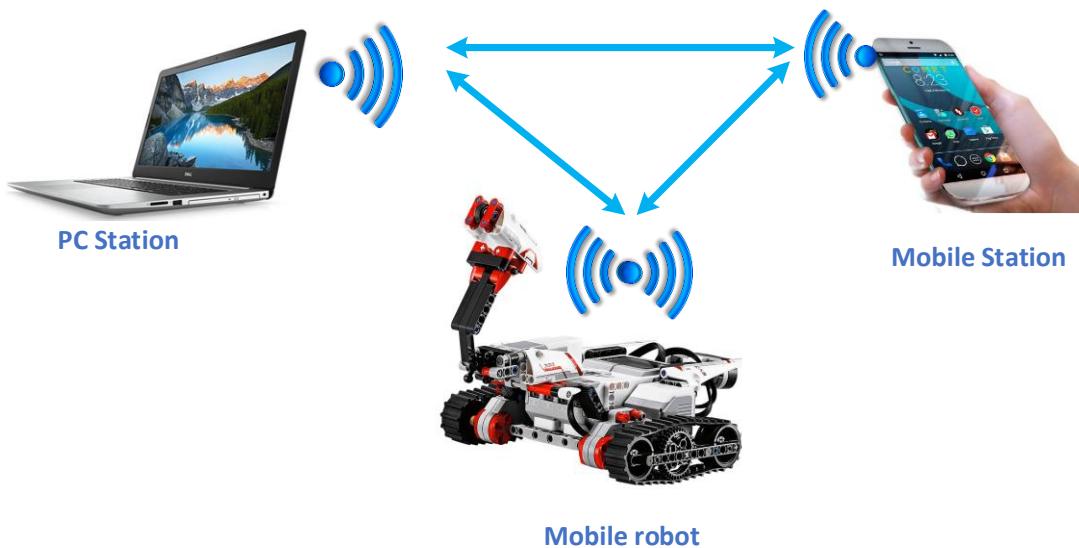
Верни отговори: 1 (а, г, д); 2 (а); 3 (а, в, г); 4 (б, г); 5 (а, б)

## Комуникация с други роботи или персонален компютър

### УПРАВЛЕНИЕ НА МОБИЛНА ПЛАТФОРМА

Управлението на мобилни роботи е сложен процес, при който се включват всички части на мобилната платформа - двигатели, сензори, комуникационни модули и други. За всеки отделен елемент или блок от робота се разработва управление в зависимост от това как искаме да управляваме системата, какви са нейните механични характеристики и хардуерни възможности.

Схемата по-долу показва как се осъществява управлението на робота. От едната страна на схемата е управляващият компютър, свързан към джойстик и модул за безжична комуникация, а от другата е роботът, който също е свързан към модул за безжична комуникация. Така реализирана схема позволява да се изпращат управляващи сигнали от компютъра към робота в едната посока, а в обратната посока да се изпращат сигнали от сензорите на робота към компютъра (обратна връзка).



*Дистанционно управление на мобилен робот*

Този тип свързване позволява да се използват възможностите на компютъра за формиране на управляващите сигнали и обработка на получените данни от робота. Това дава възможност за работа с голям обем информация, реализиране на сложни алгоритми и бързодействие. Използването на компютър редуцира работата на контролера на робота, като той само изпълнява получените команди и изпраща данните от сензорите. С това се постига по-високо бързодействието на робота, не е необходимо използването на скъпи и мощни контролери и се намалява консумираната енергия от батерията.

### БЕЗЖИЧНИ КОМУНИКАЦИОННИ ПРОТОКОЛИ

#### EEE 802.11 (Wi-Fi)

Стандартът IEEE 802.11 работи в съответствие с двете долни нива на модела OSI (Open System Interconnection) - физическо и канално ниво. Основната архитектура, особености, протоколи и служби са определени в стандарта 802.11, а спецификацията 802.11b засяга физическото ниво, променяйки скоростта на обмен и достъп до по-висока.

На физическо ниво са отделени общо три метода за предаване на данни, единият от които е в инфрачервеният диапазон, а другите два са радиочестотни, работещи в интервала между 2.4 GHz и 2.483 GHz.

Стандартът 802.11 фиксира два вида безжично мрежово оборудване - *клиент*, ролята на който обикновено се поема от компютър с инсталрирана безжична мрежова интерфейсна платка (Network Interface Card, NIC) и *точка за достъп* (Access point, AP), която служи за връзка между безжична и кабелна мрежа.

Стандартът IEEE 802.11 определя два режима на работа на безжичната мрежа - режим точка-точка (*Ad-hoc*) и режим клиент/сървър, наричан още режим на инфраструктура (*infrastructure mode*). Първият режим, точка-точка, наричан още IBSS - *независим набор от услуги*, представлява елементарна като структура мрежа, в която отделните станции се свързват една със друга пряко, без да е необходима точка за достъп.

Режимът клиент/сървър предполага използването на поне една точка за достъп, представляваща специализирано устройство, която да е включена към кабелна Ethernet мрежа и определен, често ограничен брой крайни безжични работни станции. Този тип конфигурация се нарича *основен набор от услуги* (BSS - Basic Service Set), като при наличието на два или повече BSS се формира *разширен набор от услуги* (ESS - Extended Service Set).

### IEEE 802.15.1 (Bluetooth протокол)

Bluetooth комуникациите се осъществяват в честотния диапазон 2400-2483.5 MHz. Разстояние, на което могат да се отдалечат две устройства е около 20-30 метра. Типичното разстояние обикновено не надвишава 10 метра. Няколко Bluetooth устройства могат да се свържат в мрежа и през стена (стени) или на няколко етажа в една сграда, без да има необходимост от пряка видимост или външна антена, по същия начин, по който могат да се свързват IEEE 802.11. Bluetooth има и друга, отликаваща го от останалите технологии особеност: различните Bluetooth устройства влизат в контакт едно с друго автоматично, веднага след като попаднат в обсега на приемо-предавателя, а за установяването на връзката, автентификацията и др. се грижи програмното осигуряване.

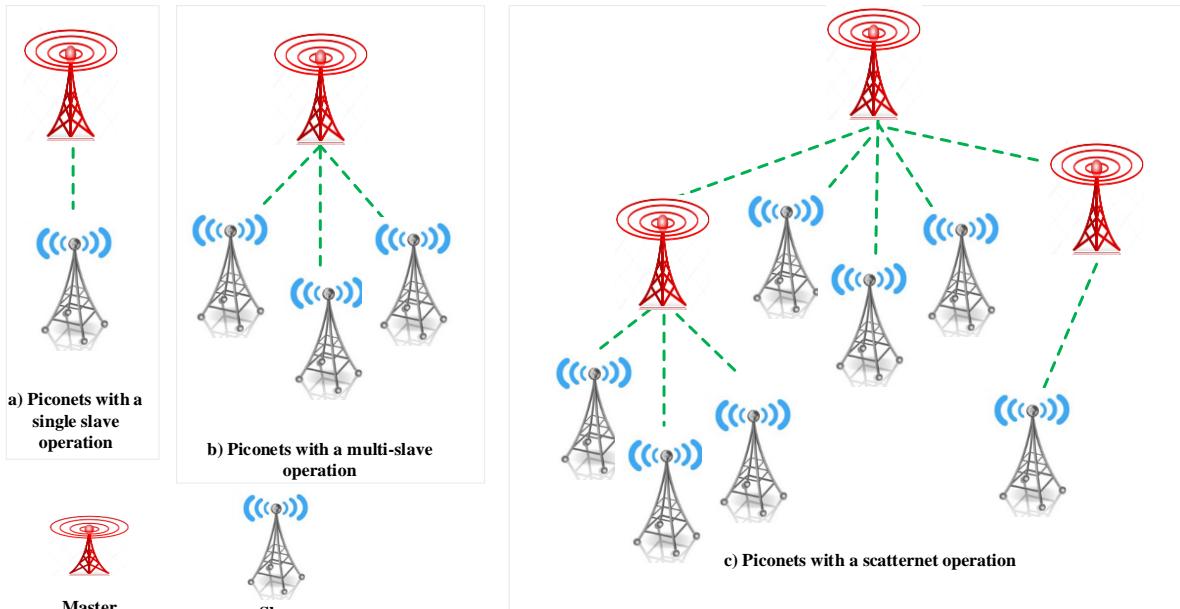
Едно от големите предимства на Bluetooth е, че устройството, поддържащо стандарта, влизайки в обхват може да установи връзка не с едно, а с множество други, поддържащи тази технология, като не е задължително те да си взаимодействват активно.

Устройство, обменящо активно информация с други устройства, според терминологията на Bluetooth се нарича *master*, а устройствата, с които то комуникира активно се наричат *slave*, като максималният брой slave устройства може да бъде 7. Освен това може да съществуват още неограничен брой неактивни slave устройства, които са установили връзка с него, макар, че са синхронизирани с master, не обменят данни с последния, очаквайки освобождаване на свободно място, за да осъществят преноса на данни. Такъв тип връзка между устройствата се нарича *piconet*. В рамките на една piconet връзка може да има само едно master устройство, но когато е необходимо, свързаното с него slave може да смени статуса си на master, образувайки своя piconet структура. Този тип сложна структура носи наименованието *scatternet*, в която всяко едно устройство може да бъде едновременно и master и slave, в зависимост от конкретната ситуация и мястото му в структурата. На фигураната по-долу е показан начинът на свързване в *piconet* с едно устройство slave (a) и с няколко (b) и свързване от типа *scatternet* (c).

Разбира се, за да се избегне дублирането на устройствата и други нежелани отклонения, всяко устройство, освен уникалното си име, взаимодейства с другите, използвайки

различен канал за връзка, на различна честота и с различен от другите параметър hopping, характеризиращ hopping channel (хопинг-канал). Хопинг (hopping)-това е периодична промяна на честотата, определяна от параметъра hopping sequence.

Bluetooth позволява икономически изгодно, непрекъснато предаване на глас и информация по безжичен път в късовълновия радиодиапазон, което дава на потребителите възможност за лесно и бързо свързване на голям брой устройства без необходимост от кабелна мрежа.



Топология на Bluetooth мрежите

### Bluetooth комуникация в Lego EV3

Една от характеристиките на Lego Mindstorms EV3 е вградената Bluetooth възможност за комуникация. Bluetooth е с обхват от около 10 метра. За да можете да използвате безжична комуникация с компютъра, той трябва да има вграден Bluetooth или присъединен Bluetooth адаптер. Има 3 основни приложения на Bluetooth в Lego Mindstorms EV3:

1. Безжична комуникация с компютър, който поддържа Bluetooth.  
Bluetooth комуникациите могат да се използват за свързване с вашия компютър, вместо към USB връзка. Можете да направите всички функции, достъпни през USB връзката чрез Bluetooth, с изключение на това, че Lego Mindstorms EV3 фърмуера може да бъде обновен само чрез USB. USB комуникациите са много по-бързи от Bluetooth, но изискват физическа връзката.
2. Безжично общувайте с други Lego Mindstorms EV3 (до три, но един по един).  
Bluetooth може да се използва за свързване на няколко робота. Когато два Lego робота са свързани чрез Bluetooth, единият играе роля на главно устройство, а другото е подчиненото устройство. Главният робот може да поддържа връзка с до три подчинени устройства. Подчиненият робот поддържа само една Bluetooth връзка.
3. Безжично комуникирайте с други устройства през Bluetooth (телефон, таблет и др.).

Можете да използвате мобилния си телефон, за да контролирате робота. Повече информация можете да намерите на адреса <https://www.lego.com/en-us/service/help/products/themes-sets/mindstorms/connecting-with-bluetooth-to-lego-mindstorms-ev3-apps-408100000007982>

#### КАК РАБОТИ ВРЪЗКАТА ГЛАВЕН / ПОДЧИНЕН

Когато няколко Lego робота са свързани заедно чрез Bluetooth комуникация, се установява връзката главен/подчинен (master/slave). Главният и подчиненият робот се държат различно. Главното устройство винаги е страна, която е създала Bluetooth връзката. Ако създадете връзка от менюто Bluetooth на Lego Mindstorms EV3, то този робот е главен.

Принципът на връзката master-slave е прост: само master може да инициира комуникация. Съобщение, генерирано от master може да пренася данни, предоставени от програмата, или може да поиска от подчинените устройства данни. Когато master изпрати данни към подчиненото устройство, той може да поиска потвърждение или не от подчиненото устройство, че данните са приети. Когато главното устройство изисква данни от подчиненото устройство, отговорът съдържа данните, ако има налични данни, или код на грешка, ако няма налични данни или ако на подчинения робот не се изпълнява програма.

Един недостатък на връзката master/slave е, че Lego Mindstorms EV3 не може да бъде едновременно главно и подчинено устройство.

Това означава, че подчиненото устройство не може просто да изпрати съобщение до главния робот. Master може да бъде в комуникационен режим с едно от другите две възможни устройства и няма да слуша този конкретен slave. Така slave трябва да "буферира" съобщението си и да изчака заявката за запитване от главния оператор с искане за буферно съобщение.

Комуникацията между Lego Mindstorms EV3 и други Bluetooth устройства се осъществява чрез един от четирите канала. Канал 0 - този канал е запазен за комуникация на подчинените устройства с главното. По подразбиране всички подчинени устройства ще изпращат данни към главния по канал 0. Другите три канала 1, 2, 3 се използват от Lego Mindstorms EV3 master за изпращане на данни или инструкции към подчинените роботи.

#### ПРИМЕРЕН КОД ЗА BLUETOOTH КОМУНИКАЦИЯ

За да демонстрираме реализирането на Bluetooth комуникация ще бъдат показани две програми, една за главния робот и една за подчинения. Тези примерни програми ще ви научат как посредством непрекъснат поток от низови съобщения може да се управляват по безжична мрежа две устройства.

Главната програма първо проверява дали подчиненото устройство е правилно свързано по линия 1 (BT\_CONN е константа), използвайки функцията BluetoothStatus (conn). След това изпраща съобщения със съдържание "Master" и нарастващ номер посредством функцията SendRemoteString (conn, queue, string). Чрез функцията ReceiveRemoteString (queue, clear, string) се получават съобщения от подчиненото устройство, които се показват на екрана.



```
//MASTER
#define BT_CONN 1 #define INBOX 1 #define OUTBOX 5
sub BTCheck(int conn){
    if (!BluetoothStatus(conn)==NO_ERR) {
        TextOut(5,LCD_LINE2,"Error");
        Wait(1000);
        Stop(true);
    }
}
task main(){
    string in, out, iStr;
    int i = 0;
    BTCheck(BT_CONN); //check slave connection
    while(true){
        iStr = NumToStr(i);
        out = StrCat("Master",iStr);
        TextOut(10,LCD_LINE1,"Master Test");
        TextOut(0,LCD_LINE2,"IN:");
        TextOut(0,LCD_LINE4,"OUT:");
        ReceiveRemoteString(INBOX, true, in);
        SendRemoteString(BT_CONN,OUTBOX,out);
        TextOut(10,LCD_LINE3,in);
        TextOut(10,LCD_LINE5,out);
        Wait(100);
        i++;
    }
}
```

Подчинената програма е много подобна, но използва функцията SendResponseString (queue,string) вместо SendRemoteString, защото подчиненото устройство може да изпраща съобщения само на своя главен робот по линия 0.



```

//SLAVE
#define BT_CONN 1
#define INBOX 5
#define OUTBOX 1
sub BTCheck(int conn){
    if (!BluetoothStatus(conn)==NO_ERR) {
        TextOut(5,LCD_LINE2,"Error");
        Wait(1000);
        Stop(true);
    }
}
task main(){
    string in, out, iStr;
    int i = 0;
    BTCheck(0); //check master connection
    while(true){
        iStr = NumToStr(i); out = StrCat("S",iStr);
        TextOut(10,LCD_LINE1,"Slave Test");
        TextOut(0,LCD_LINE2,"IN:");
        TextOut(0,LCD_LINE4,"OUT:");
        ReceiveRemoteString(INBOX, true, in);
        SendResponseString(OUTBOX,out);
        TextOut(10,LCD_LINE3,in);
        TextOut(10,LCD_LINE5,out);
        Wait(100); i++;
    }
}

```

Ще забележите, че ако спре една от програмите, другата ще продължи да изпраща съобщения с нарастващ номер, без да знае, че всички изпратени съобщения ще бъдат загубени, защото никой няма да слуша от другата страна. За да избегнем този проблем, можем да използваме по-добър протокол за връзка очакващ потвърждение за доставка на данните.

#### ИЗПРАЩАНЕ НА СЪОБЩЕНИЕ С ИЗЧАКВАНЕ НА ПОТВЪРЖДЕНИЕ

В следващите примери главното устройство изпраща съобщението към подчиненото чрез функцията `SendRemoteNumber(conn, queue, number)` и спира за да изчака получаването на потвърждение (реализира се с цикъла, в който се намира функцията `ReceiveRemoteString()`). Главното устройство ще продължи да изпраща данни, само ако подчиненото отговори на съобщението. Подчиненото устройство получава съобщения с функцията `ReceiveRemoteNumber(queue,clear,number)` и изпраща ack с `SendResponseNumber`.

Двете програми (master и slave) използват един код за потвърждение `ack = 0xFF`.

Master изпраща случаини числа и изчаква отговор от подчиненото. Когато получи ack с правилния код, той се изчиства.

Подчиненият проверява непрекъснато пощенската кутия и, ако не е празна, показва прочетената стойност и изпраща ack на master. В началото на програмата се изпраща ack съобщение, за да отблокира master.



```

//MASTER
#define BT_CONN 1
#define OUTBOX 5
#define INBOX 1
#define CLEARLINE(L) \
    TextOut(0,L,"    ");
sub BTCheck(int conn){
    if (!BluetoothStatus(conn)==NO_ERR) {
        TextOut(5,LCD_LINE2,"Error");
        Wait(1000);
        Stop(true);
    }
}

task main(){
    int ack;
    int i;
    BTCheck(BT_CONN);
    TextOut(10,LCD_LINE1,"Master sending");
    while(true){
        i = Random(512);
        CLEARLINE(LCD_LINE3);
        NumOut(5,LCD_LINE3,i); ack = 0;
        SendRemoteNumber(BT_CONN,OUTBOX,i);
        until(ack==0xFF) {
            until(ReceiveRemoteNumber(INBOX,true,ack) == NO_ERR);
        }
        Wait(250);
    }
}

//SLAVE
#define BT_CONN 1
#define OUT_MBOX 1
#define IN_MBOX 5
sub BTCheck(int conn){
    if (!BluetoothStatus(conn)==NO_ERR) {
        TextOut(5,LCD_LINE2,"Error");
        Wait(1000);
        Stop(true);
    }
}
task main(){
    int in;
    BTCheck(0);
    TextOut(5,LCD_LINE1,"Slave receiving");
    SendResponseNumber(OUT_MBOX,0xFF); //unblock master
    while(true){
        if (ReceiveRemoteNumber(IN_MBOX, true,in) != STAT_MSG_EMPTY_MAILBOX) {
            TextOut(0,LCD_LINE3,"    ");
            NumOut(5,LCD_LINE3,in);
            SendResponseNumber(OUT_MBOX,0xFF);
        }
        Wait(10); //take breath (optional)
    }
}

```

Има и друга функция за Bluetooth комуникацията. Чрез нея master може директно да контролира своите подчинени устройства. В следващия пример master изпраща към подчинените роботи директни команди за възпроизвеждане на звуци и задвижване на



двигателите. Тук не се използва подчинена програма, тъй като фърмуерът на подчинения Lego Mindstorms EV3 получава и управлява съобщенията.

```
//MASTER
#define BT_CONN 1
#define MOTOR(p,s) RemoteSetOutputState(BT_CONN, p, s, \
    OUT_MODE_MOTORON+OUT_MODE_BRAKE+OUT_MODE_REGULATED, \
    OUT_REGMODE_SPEED, 0, OUT_RUNSTATE_RUNNING, 0)

sub BTCheck(int conn){
    if (!BluetoothStatus(conn)==NO_ERR){
        TextOut(5,LCD_LINE2,"Error");
        Wait(1000);
        Stop(true);
    }
}
task main(){
    BTCheck(BT_CONN);
    RemotePlayTone(BT_CONN, 4000, 100);
    until(BluetoothStatus(BT_CONN)==NO_ERR);
    Wait(110);
    RemotePlaySoundFile(BT_CONN, "!Click.rso", false);
    until(BluetoothStatus(BT_CONN)==NO_ERR);
    //Wait(500);
    RemoteResetMotorPosition(BT_CONN,OUT_A,true);
    until(BluetoothStatus(BT_CONN)==NO_ERR);
    MOTOR(OUT_A,100);
    Wait(1000);
    MOTOR(OUT_A,0);
}
```

#### IEEE 802.15.4 (ZigBee)

ZigBee технологията е разработена основно за нуждите от евтино решение, базирано на стандарта за безжични мрежи, което да поддържа малки потоци от данни, ниска консумация и надеждност. Това е единствената до момента стандартизирана технология, насочена към уникалните нужди на мрежови приложения, свързани с дистанционно наблюдение и контрол на различни сензори. ZigBee технологията е подходяща за широк клас от приложения в много области.

Едно от основните предимства на устройствата, ползвани с ZigBee технологията е ниската консумация, което от своя страна, дава възможност за многократно увеличаване живота на батерии. Устройствата, използвани батерии като захранващ източник, могат да се установят в неактивен режим (sleep).

За разлика от Bluetooth, който има различни режими и състояния, ZigBee има два основни режима - активен (изпраща/приема) и пасивен ("sleep"). Приложният софтуер е необходимо да се фокусира върху самото приложение, а не върху избора на оптимален захранващ режим за всеки различен случай.

Основните технически характеристики на протокол 802.15.04/ZigBee са:

- Потокът от данни 250Kbs, може да се постигне при 2.4GHz (10 канала), 40Kbs при 915Mhz (16 канала) и 20Kbs при 868Mhz (1 канал);
- Разстоянието за предаване е от 10 до 75 м в зависимост от мощността на захранването и особеностите на терена;
- Връзката между възлите може да бъде тип "звезда" или "точка до точка" (Peer-to-Peer);



- Много ниска консумация (под 1 mV);

Подобно на WiFi и Bluetooth и ZigBee също работи в долните две нива (физическо и канално) на общиия модел за междусистемни връзки (OSI - Open System Interconnection).

#### СРАВНИТЕЛНИ ХАРАКТЕРИСТИКИ НА БЕЗЖИЧНИТЕ ПРОТОКОЛИ

Сравнение между безжичните протоколи в буквалния смисъл на думата не може да бъде направено, но може да се даде класификация, според спецификата на съответния използван протокол. Също така не може да се направи разграничението кой протокол е по-добър, заради различната им насоченост. Може да се направи, обаче паралел, според който да се определи кой протокол в коя област е най-подходящо да се използва.

Технологии, базирани на протокол ZigBee се използват там, където има мрежи от сензори, контрол на данни и апаратура и където се налага консумацията да е минимална. Много фирми предлагат решения за вграждане в бита и развитие на т. нар. "умно жилище", където интерфейсът се използва за връзка с всички уреди и до централен команден център.

Bluetooth е протокол, който масово се използва при мобилните телефони, а напоследък и при компютърните системи, когато е необходимо да се премахнат кабелите - безжични клавиатури, мишки и др. Правят се и разработки за вграждането на тази технология в битовата техника.

Протоколът 802.11 или наричан още Wi-Fi основно се прилага за изграждане на безжични компютърни мрежи. Най-простият начин за свързване на два компютъра в безжична мрежа е да се включи във всеки един от тях по една WLAN карта, да се инсталират драйверите и да се настройт необходимите параметри на връзка. В таблица 10.1 е дадена сравнителна характеристика на безжичните протоколи.

Таблица 4: Сравнителна характеристика на безжични протоколи

Протокол Характеристика	ZigBee™ 802.15.4	Bluetooth™ 802.15.1	Wi-Fi™ 802.11b
Приложение	Наблюдение и контрол	Заместване на кабелите	Web, Video, Email
Системни ресурси	4KB-32KB	250KB	1MB
Живот на батерията (дни)	100-1000+	1-7	1-5
Възли в мрежа	255/65K+	7	30
Честотна лента(kbps)	20-250	720	11,000+
Обхват (метри)	1-75+	1-10+	1-100
Ключови характеристики	Стабилност, Ниска консумация, Евтино	Цена, Удобство	Скорост, Гъвкавост

**ОЦЕНКА НА ЗНАНИЯТА**

1. Най-често Bluetooth мрежите са master-slave, като максималния брой slave устройства при Lego може да бъде:

- a) 3
- б) 5
- в) 7
- г) Няма ограничение

2. Колко топологии на Bluetooth мрежите съществуват?

- a) 1
- б) 2
- в) 3
- г) Липсват за специални за мрежова топология.

3. Коя от мрежите има най-голям обхват на покритие?

- a) Bluetooth
- б) ZigBee
- в) Wi-Fi
- г) USB

4. Коя от мрежите е най-икономична по отношение на консумирана енергия?

- a) Bluetooth
- б) ZigBee
- в) Wi-Fi
- г) USB

5. Кои от интерфейсите са вградени в Lego Mindstorms EV3?

- a) Bluetooth
- б) ZigBee
- в) Wi-Fi
- г) USB

*Верни отговори: 1 (а); 2(б); 3(б); 4(б); 5(а)*