

ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ ФУНКЦИОНАЛЬНОЕ УПРАВЛЕНИЕ РОБОТОМ

Нами уже были разобраны реализации циклов, вложенных циклов и структуры множественного выбора в языке С. Все это позволило программировать управление движением робота с пульта управления с использованием одной, двух или четырех кнопок.

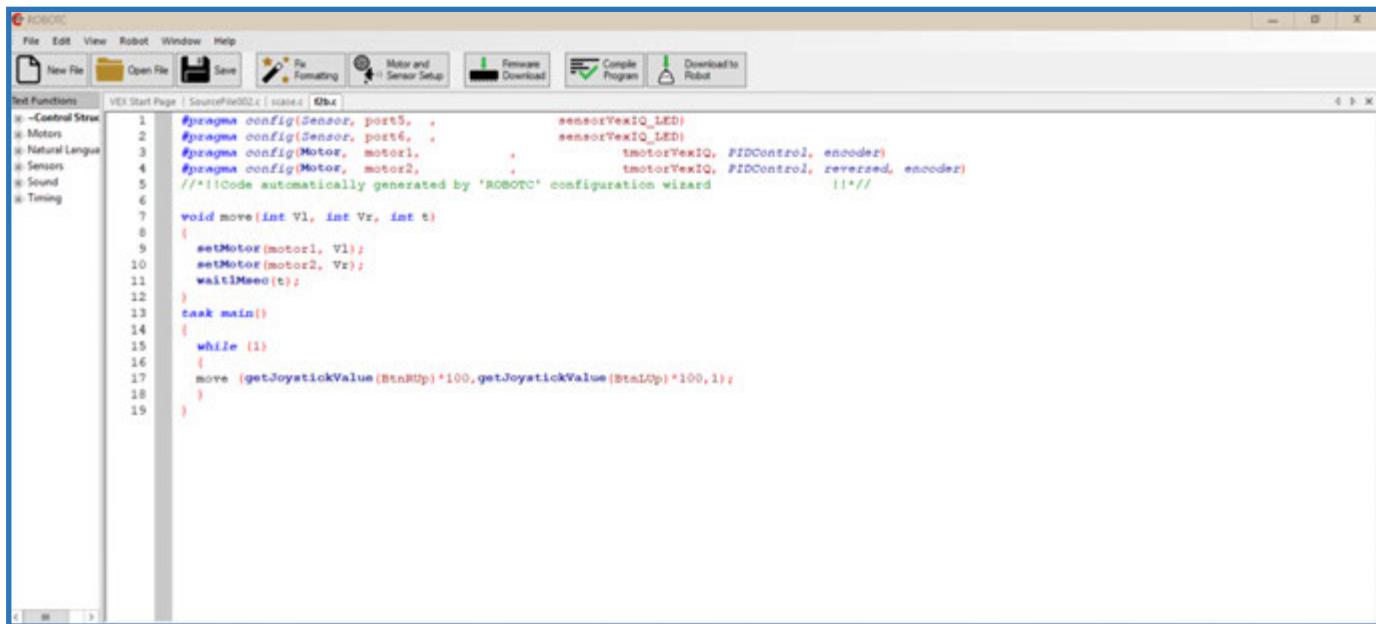
Настало время осуществить новый метод управления роботом с пульта. Назовем это метод функциональным управлением. То есть нам необходимо создать функцию, которая в зависимости от комбинации нажатых кнопок выдавала бы такие значения для каждого из колес: **-100, 0, 100**.

В этом случае имеется возможность «вставить» эту функцию в функцию **move()** в качестве первого и второго аргументов для левой и правой пары колес соответственно.

Но для начала рассмотрим более простой вариант. Реализуем функцию для двух кнопок. Она проста:

- Скорость левых колес = **getJoystickValue(BtnRUp) * 100**
- Скорость правых колес = **getJoystickValue(BtnLUp) * 100**

При нажатии на любую из кнопок соответствующая пара колес движется вперед, при отпускании колеса останавливаются.



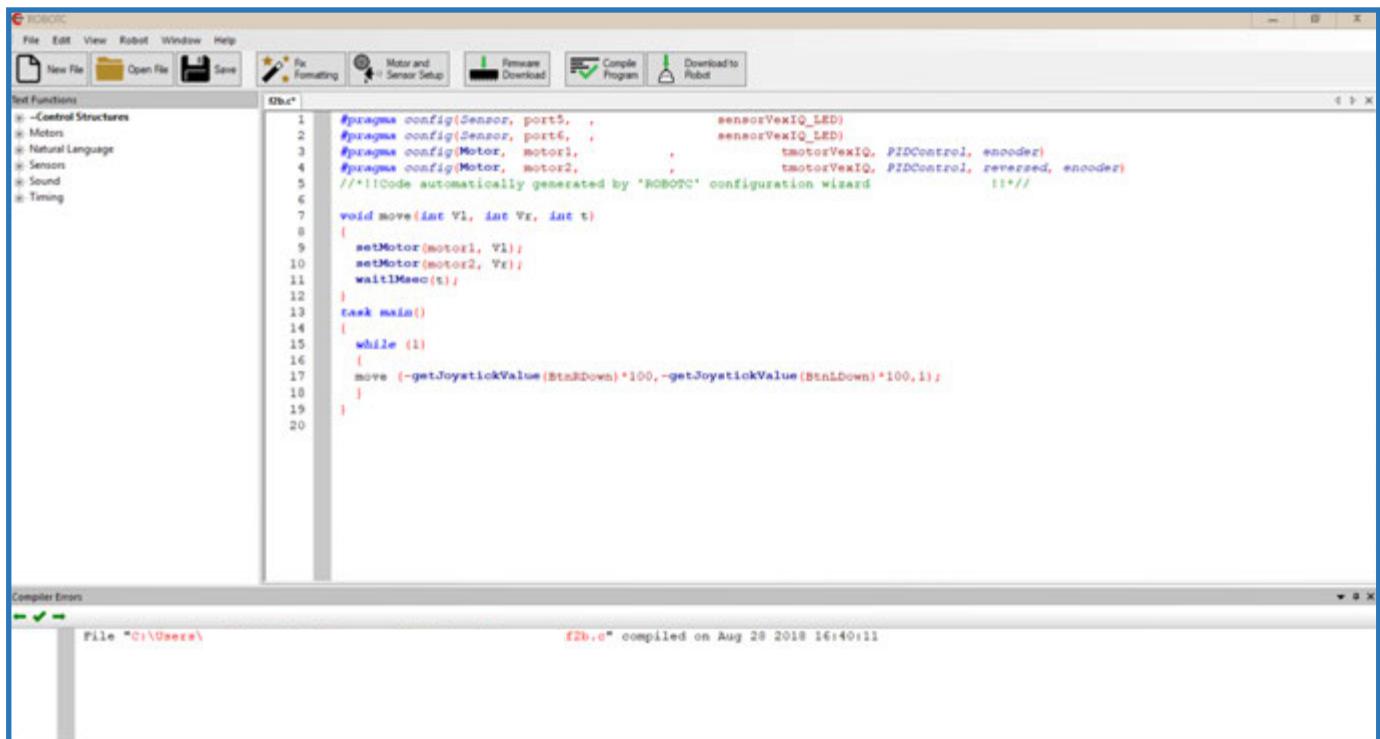
```

File Edit View Robot Window Help
New File Open File Save Format Motor and Sensor Setup Remove Download Compile Program Download to Robot
Ref Functions VEX Start Page SourceFile002.c [source] 684
--Control Struct
Motors Natural Language Sensors Sound Timing
1 #pragma config(Sensor, port5, , sensorVexIQ_LED)
2 #pragma config(Sensor, port6, , sensorVexIQ_LED)
3 #pragma config(Motor, motor1, , tmotorVexIQ, PIDControl, encoder)
4 #pragma config(Motor, motor2, , tmotorVexIQ, PIDControl, reversed, encoder)
//Code automatically generated by 'ROBOTC' configuration wizard !!
5
6
7 void move(int Vl, int Vr, int t)
8 {
9     setMotor(motor1, Vl);
10    setMotor(motor2, Vr);
11    waitIMsec(t);
12 }
13 task main()
14 {
15     while (1)
16     {
17         move (getJoystickValue(BtnRUp)*100, getJoystickValue(BtnLUp)*100, 1);
18     }
19 }

```

Все четыре состояния реализованы, робот ездит, стоит на месте, разворачивается по часовой и против часовой стрелок.

Аналогичным образом будет реализованы и движения назад:



The screenshot shows the ROBOTC IDE interface. The main window displays the following C-like pseudocode:

```
#pragma config(Sensor, port5, , sensorVexIQ_LED)
#pragma config(Sensor, port6, , sensorVexIQ_LED)
#pragma config(Motor, motor1, , tmotorVexIQ, PIDControl, encoder)
#pragma config(Motor, motor2, , tmotorVexIQ, PIDControl, reversed, encoder)
//Code automatically generated by 'ROBOTC' configuration wizard //11

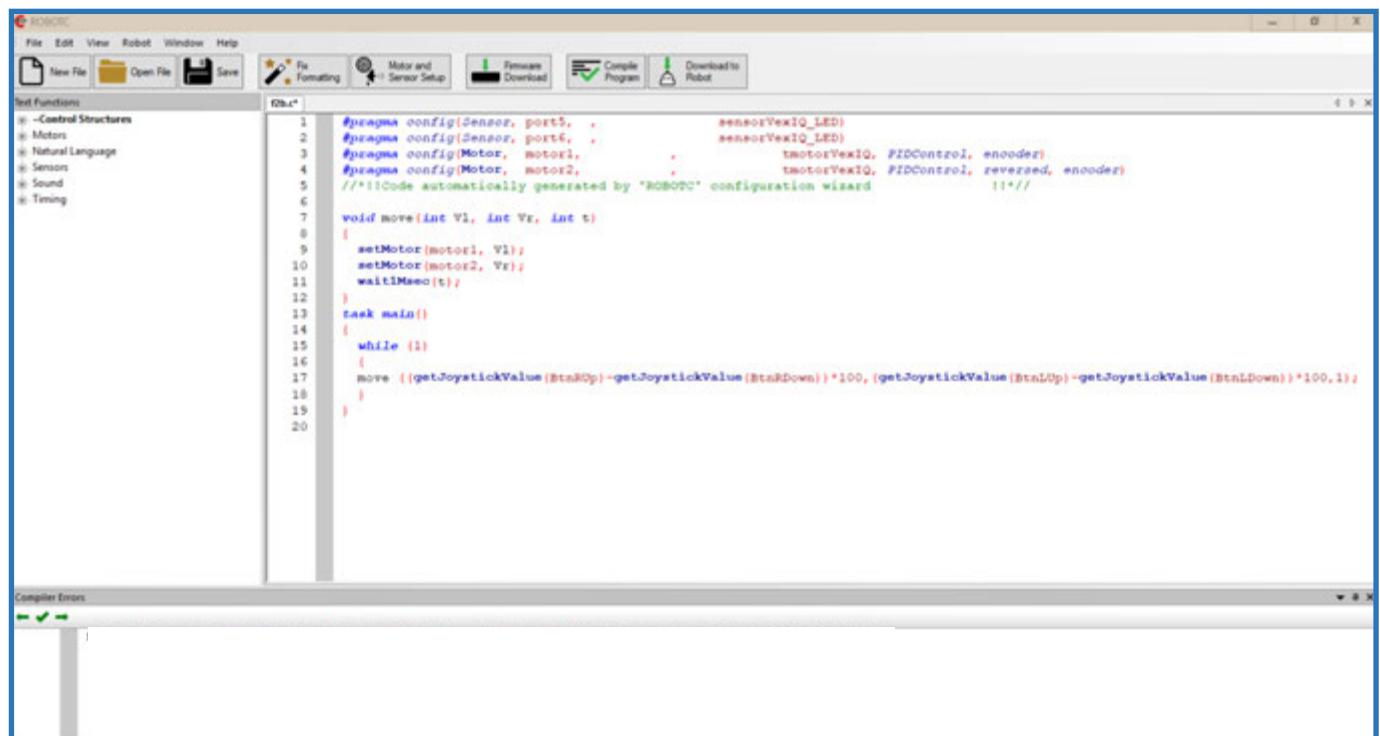
void move(int Vl, int Vr, int t)
{
    setMotor(motor1, Vl);
    setMotor(motor2, Vr);
    wait1Msec(t);
}

task main()
{
    while (1)
    {
        move (-getJoystickValue(BtnRDown)*100,-getJoystickValue(BtnLDown)*100,1);
    }
}
```

The code defines a function `move` that takes three parameters: `Vl`, `Vr`, and `t`. It uses the `setMotor` command to set the speeds of two motors. The `task main()` loop calls the `move` function with specific values for the joystick buttons. The file was compiled on Aug 28 2018 16:40:11.

За маневры задним ходом отвечают нижние курки, а скорости должны быть отрицательными, то есть **-100**.

Осталось объединить два указанных подхода:



The screenshot shows the ROBOTC IDE interface with the same code as the previous screenshot, but with additional logic added to handle the right joystick button. The code now includes a condition to switch between forward and reverse movement based on the right joystick button state:

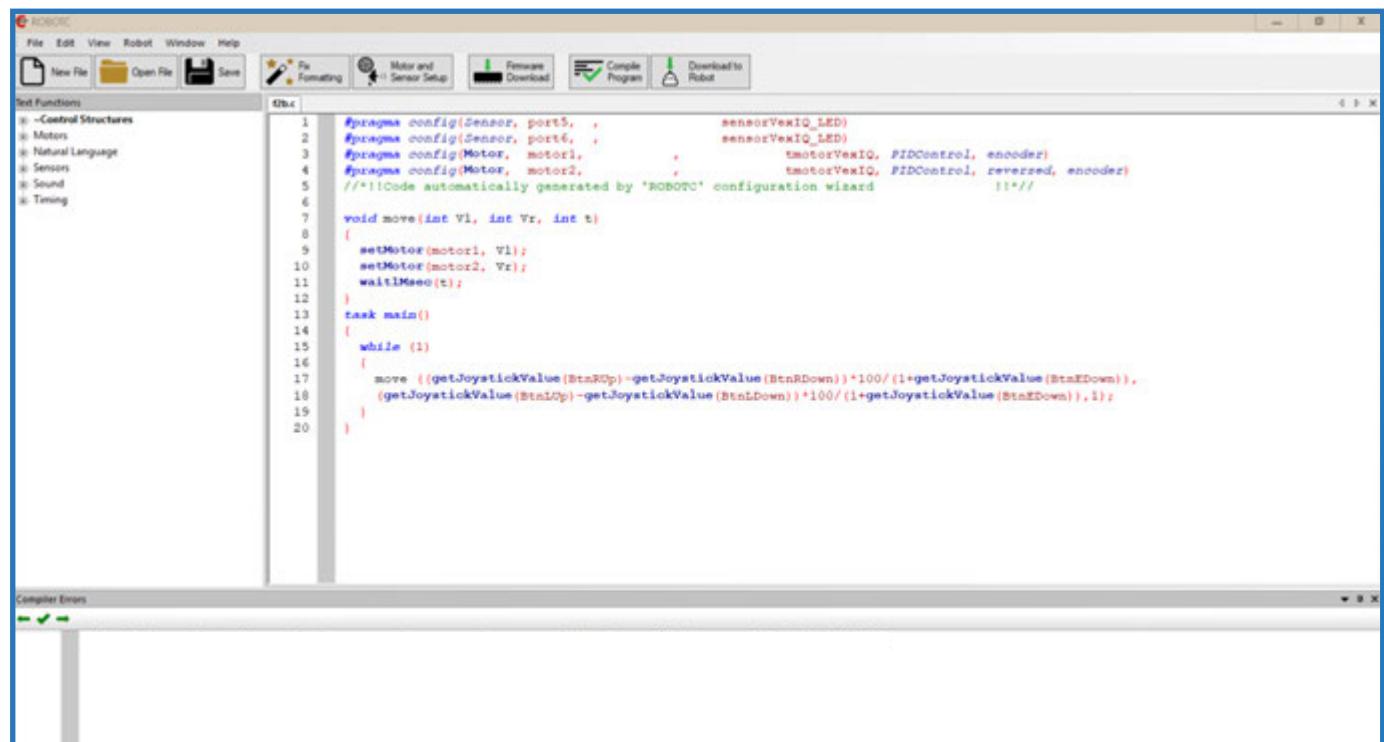
```
#pragma config(Sensor, port5, , sensorVexIQ_LED)
#pragma config(Sensor, port6, , sensorVexIQ_LED)
#pragma config(Motor, motor1, , tmotorVexIQ, PIDControl, encoder)
#pragma config(Motor, motor2, , tmotorVexIQ, PIDControl, reversed, encoder)
//Code automatically generated by 'ROBOTC' configuration wizard //11

void move(int Vl, int Vr, int t)
{
    if (getJoystickValue(BtnRUp) > getJoystickValue(BtnRDown))
        setMotor(motor1, Vl);
    else
        setMotor(motor1, -Vl);
    if (getJoystickValue(BtnLUp) > getJoystickValue(BtnLDown))
        setMotor(motor2, Vr);
    else
        setMotor(motor2, -Vr);
    wait1Msec(t);
}

task main()
{
    while (1)
    {
        move ((getJoystickValue(BtnRUp)-getJoystickValue(BtnRDown))*100, (getJoystickValue(BtnLUp)-getJoystickValue(BtnLDown))*100,1);
    }
}
```

Нами получена универсальная функция, которая реализует все возможные варианты движения робота с использованием 4 кнопок. Но и эту функцию можно улучшить.

Давайте представим, что в одних ситуациях работу нужно перемещаться, как можно быстрее, а в каких-то совершать медленные, но точные движения. Для этого робота надо «замедлять», если одна из не занятых кнопок нажата. Пусть это будет кнопка **BtnEDown**.



The screenshot shows the ROBOCOP software interface. The main window has a toolbar at the top with icons for File, Edit, View, Robot, Window, Help, and various tools like Reformatting, Motor and Sensor Setup, Firmware Download, Compile Program, and Download to Robot. Below the toolbar is a sidebar titled 'Ref Functions' containing sections for Control Structures, Motors, Natural Language, Sensors, Sound, and Timing. The main code editor area contains the following C-like pseudocode:

```
#pragma config(Sensor, port5, , sensorVexIQ_LED)
#pragma config(Sensor, port6, , sensorVexIQ_LED)
#pragma config(Motor, motor1, , tmotorVexIQ, PIDControl, encoder)
#pragma config(Motor, motor2, , tmotorVexIQ, PIDControl, reversed, encoder)
//**!Code automatically generated by 'ROBOTC' configuration wizard */

void move(int Vl, int Vr, int t)
{
    setMotor(motor1, Vl);
    setMotor(motor2, Vr);
    wait1Msec(t);
}

task main()
{
    while (1)
    {
        move ((getJoystickValue(BtnRUp)-getJoystickValue(BtnEDown))*100/(1+getJoystickValue(BtnEDown)), 
               (getJoystickValue(BtnLUp)-getJoystickValue(BtnLDown))*100/(1+getJoystickValue(BtnEDown)), 1);
    }
}
```

The code is intended to control two motors based on joystick inputs. It includes logic to slow down movement if the BtnEDown button is pressed.

Из-за того, что код не влезал в одну строку 17 строка была разбита на две строки. Это не приводит к какой-либо ошибке потому что компилятор концом строки считает **;**. Модернизация же для контроля скорости состоит в том, что в 17-18 строке добавлено деление на **(1 + getJoystickValue(BtnEDown))**. Эта скобка равна **1**, если кнопка не нажата, и равна **2**, если – нажата. Модернизировав эту скобку, добавив коэффициент перед **getJoystickValue(BtnEDown)** можно добиться большего замедления. Не забывайте, что коэффициент должен быть больше единицы и быть целым числом.