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 движение робота с пульта управления с использованием одной, двух или четырех кнопок.

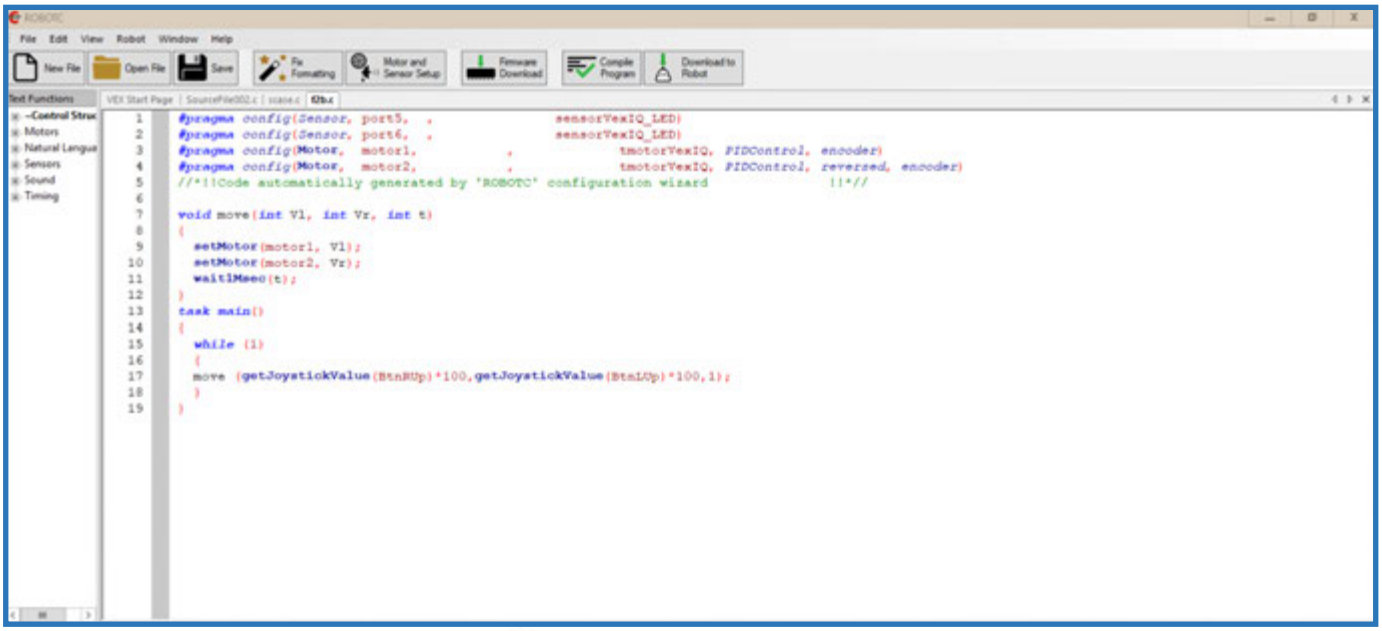
Настало время осуществить новый метод управления роботом с пульта. Назовем это метод функциональным управлением. То есть нам необходимо создать функцию, которая в зависимости от комбинации нажатых кнопок выдавала бы такие значения для каждого из колес: `-100`, `0`, `100`.

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:

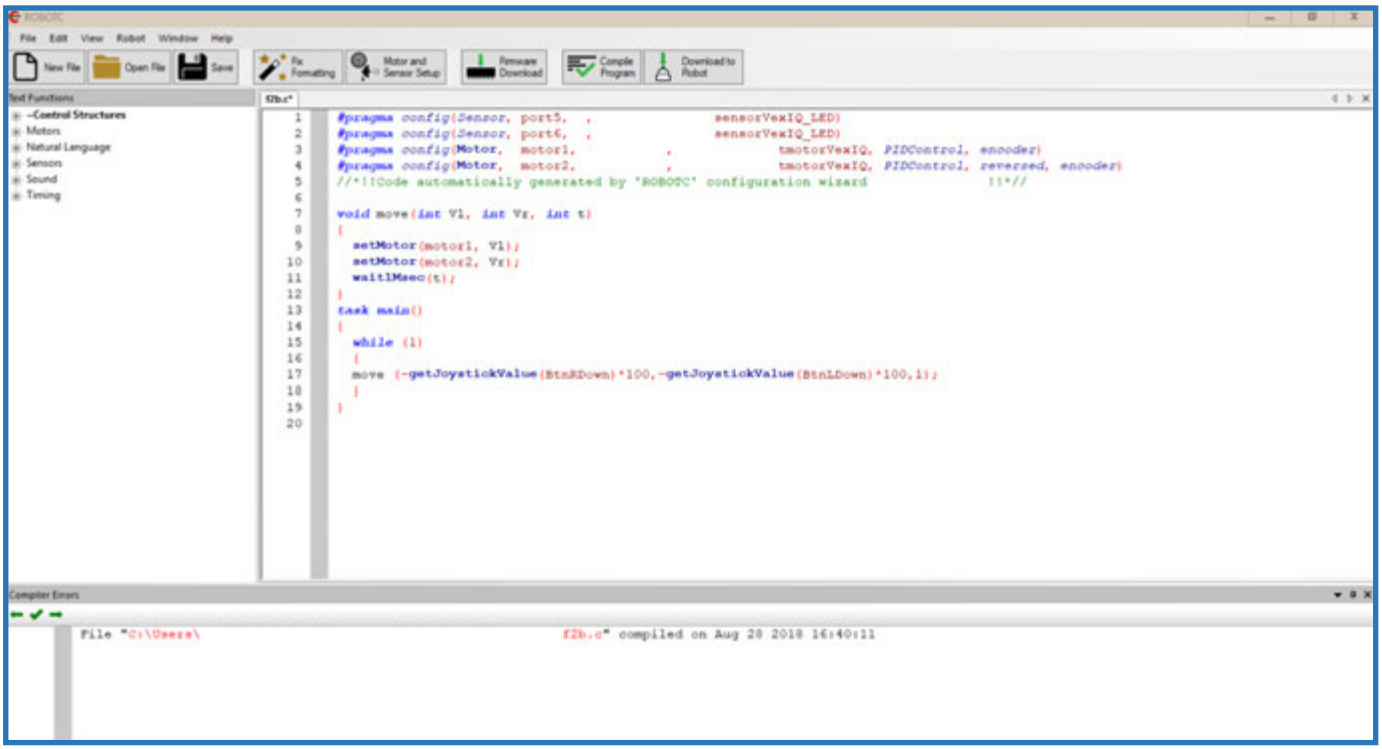
- Скорость левых колес = `getJoystickValue(BtnRUp)* 100`
- Скорость правых колес = `getJoystickValue(BtnLUp)* 100`

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



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

Аналогичным образом будет реализованы и движения назад:



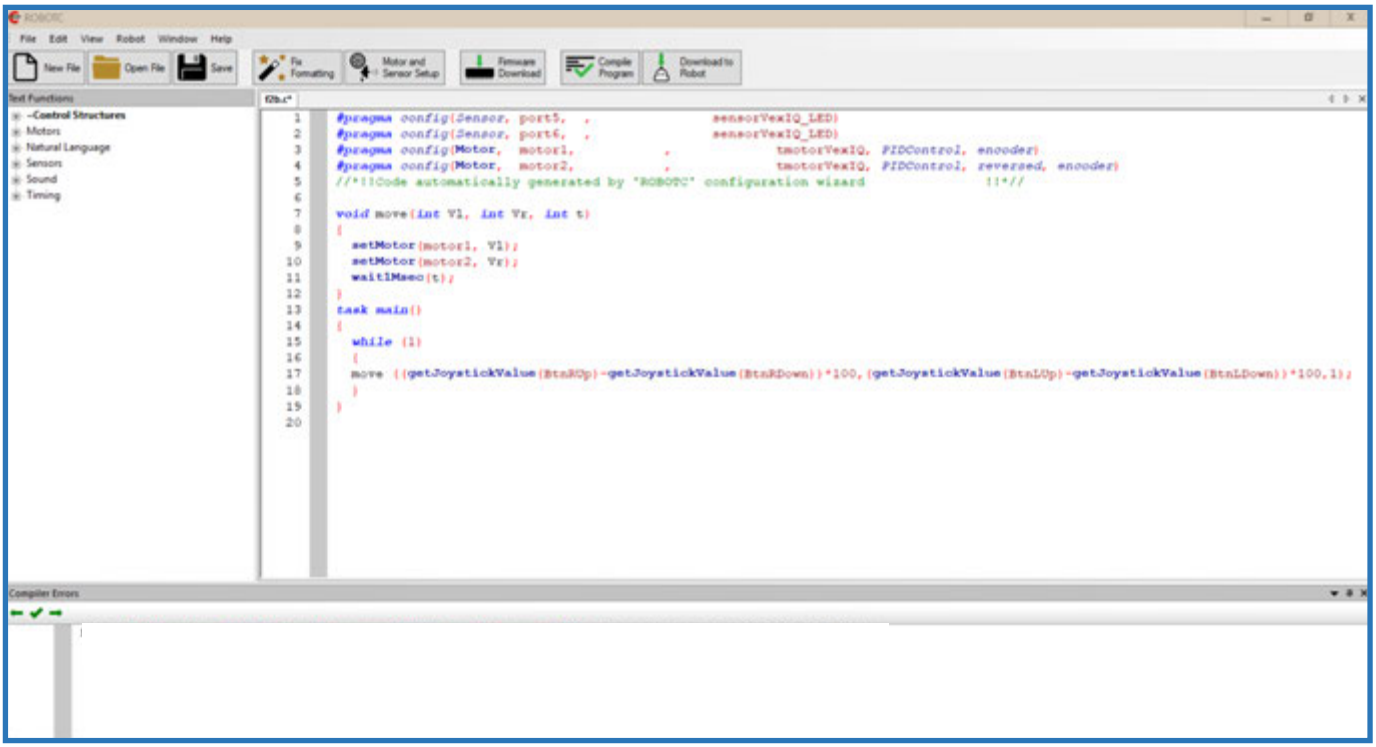
```
#pragma config(Sensor, port5,  ,               sensorVexIQ_LED)
#pragma config(Sensor, port6,  ,               sensorVexIQ_LED)
#pragma config(Motor,  motor1,           ,             tmotorVexIQ, PIDControl, encoder)
#pragma config(Motor,  motor2,           ,             tmotorVexIQ, PIDControl, 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(BtnRDown)*100,-getJoystickValue(BtnLDown)*100,1);
  }
}
```

```
File "C:\Users\                    f2b.c" 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:

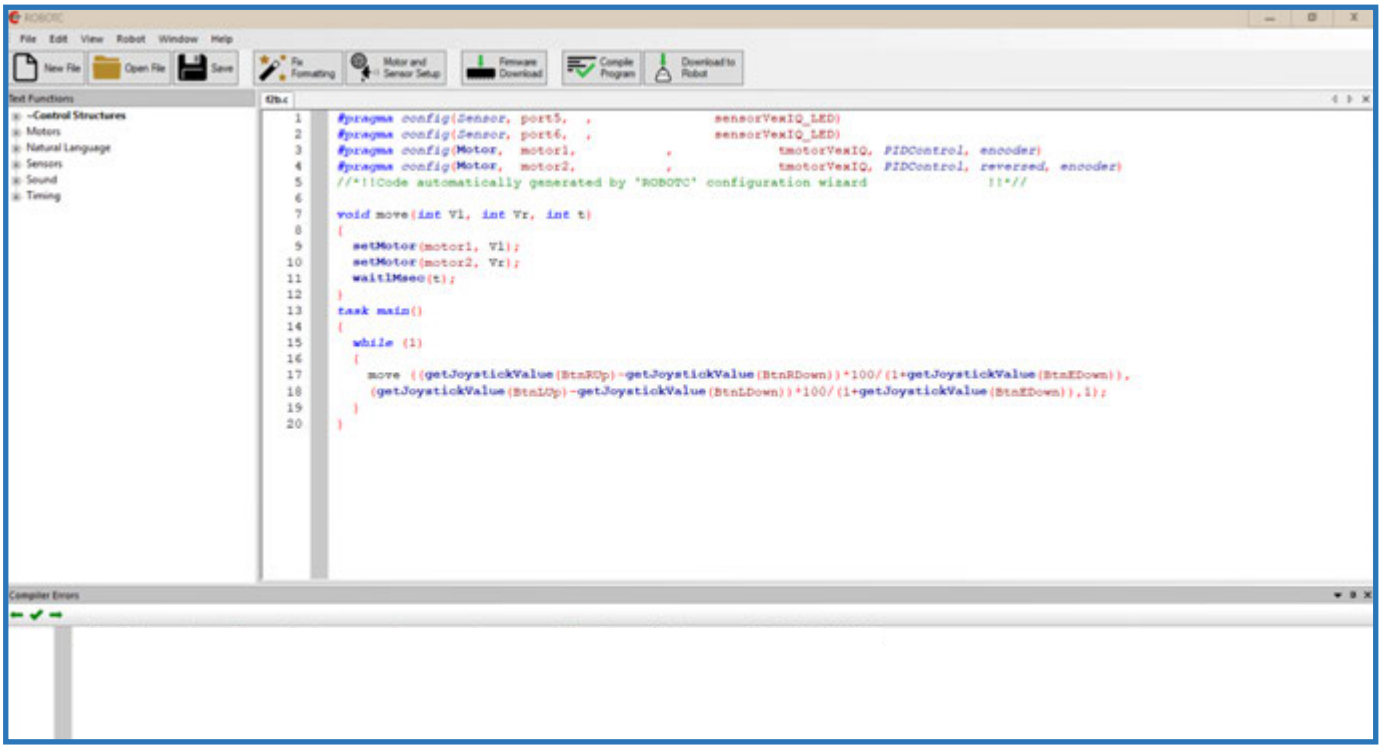


```
#pragma config(Sensor, port5,  ,               sensorVexIQ_LED)
#pragma config(Sensor, port6,  ,               sensorVexIQ_LED)
#pragma config(Motor,  motor1,           ,             tmotorVexIQ, PIDControl, encoder)
#pragma config(Motor,  motor2,           ,             tmotorVexIQ, PIDControl, 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(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 роботу 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`.



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.