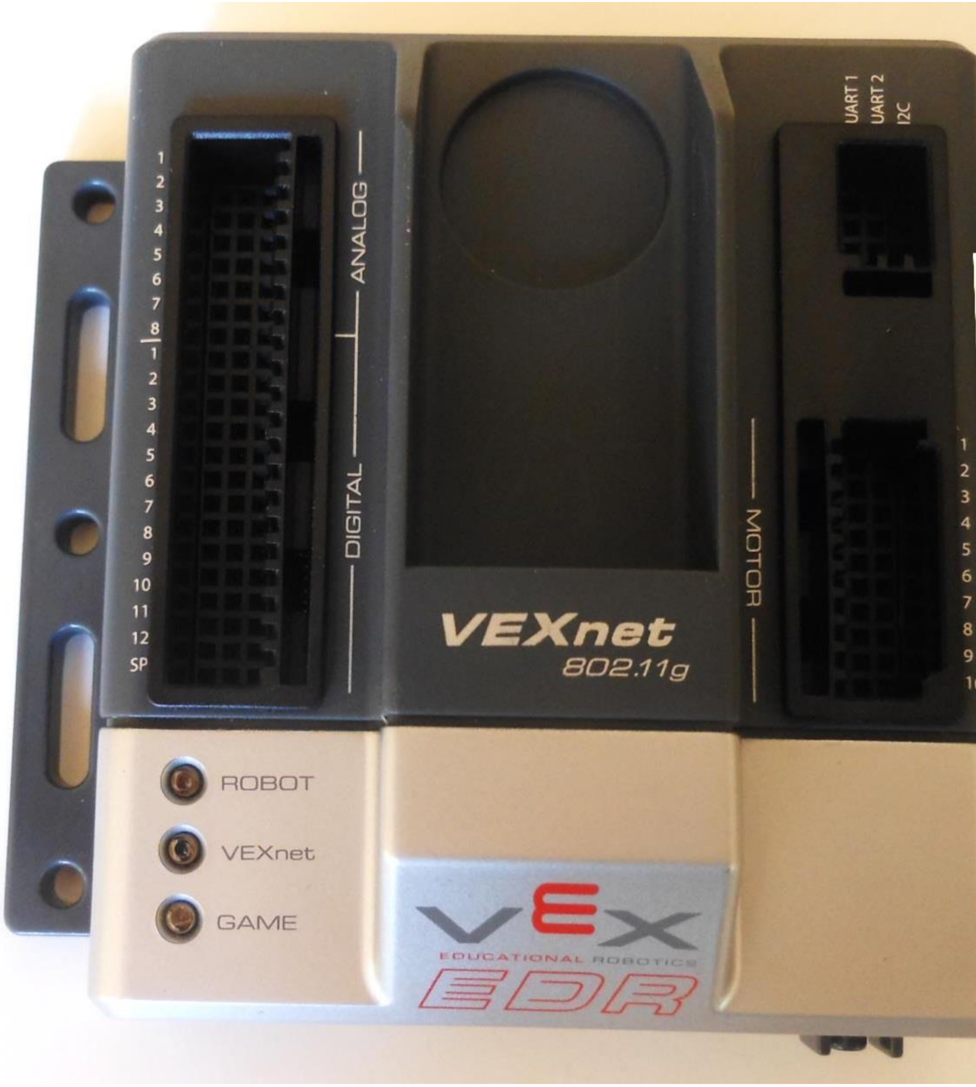
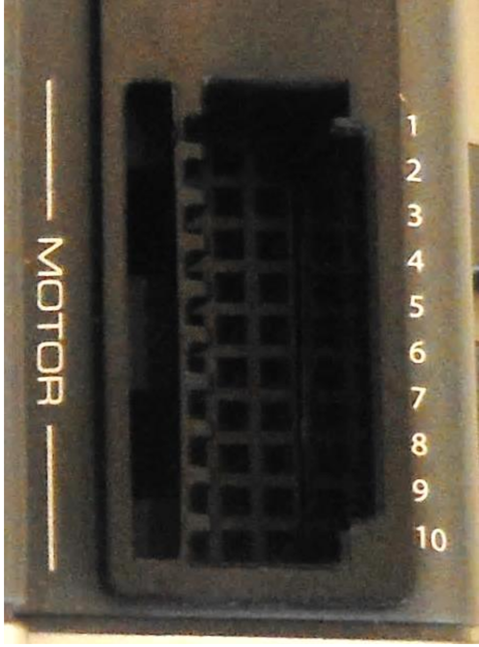
Mechanical Components and Programming

7.2 V, 3000 mAh battery pack and charger

Controller (Cortex)

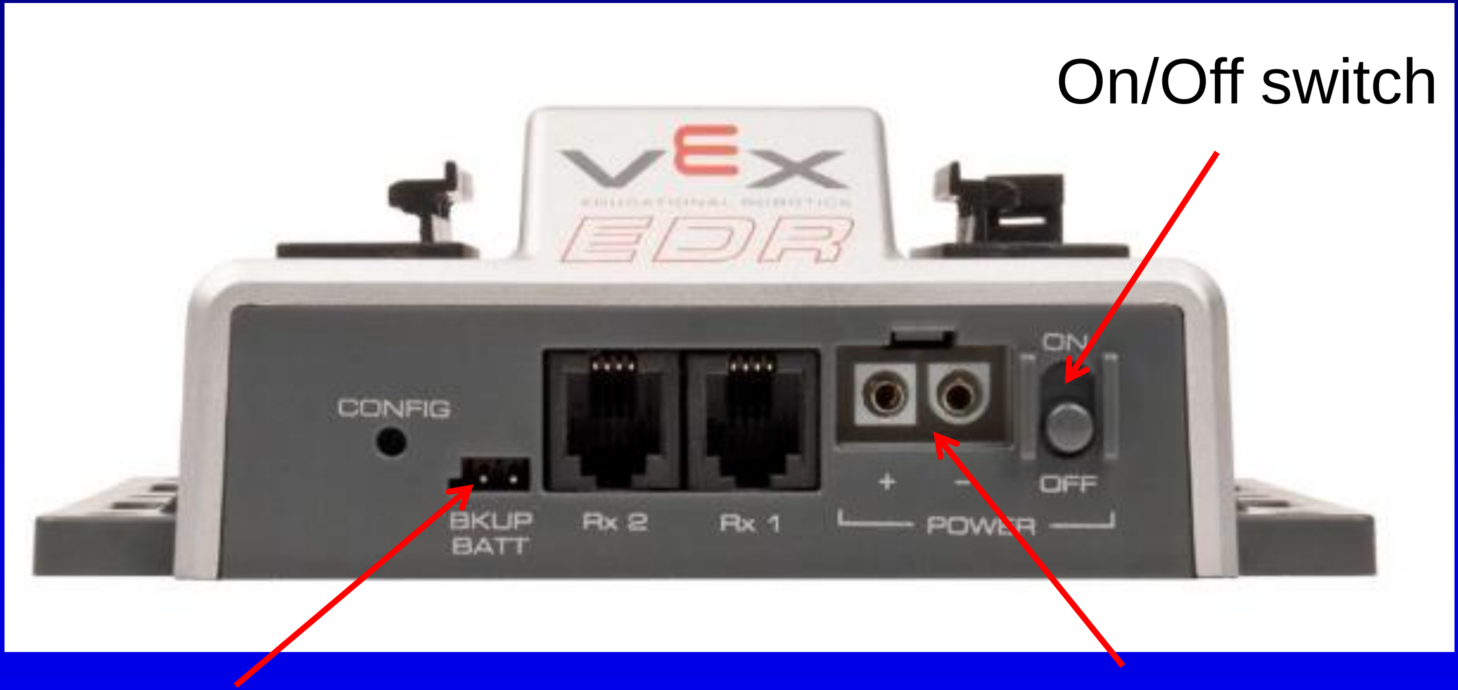


Motor and Servo ports



8 analog and 12 digital ports. Digital ports are used for Limit, Bumper and Ultrasonic sensors

Side view of the VEX Controller

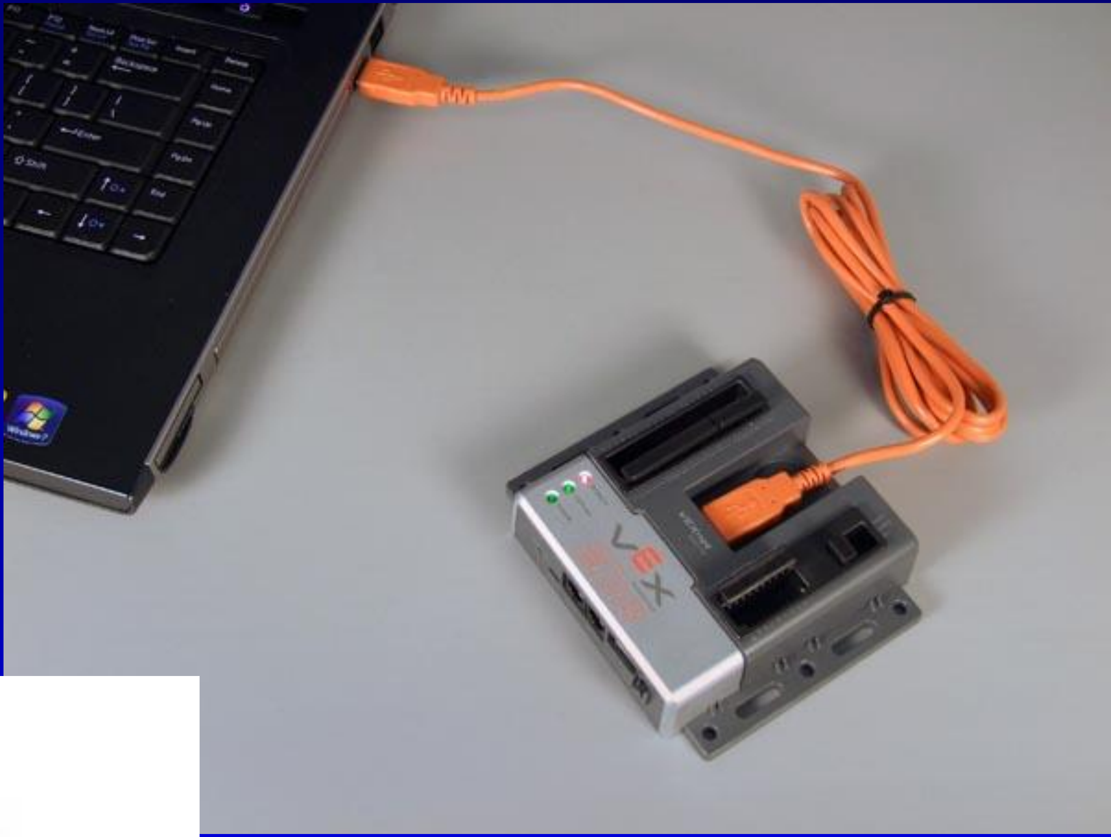


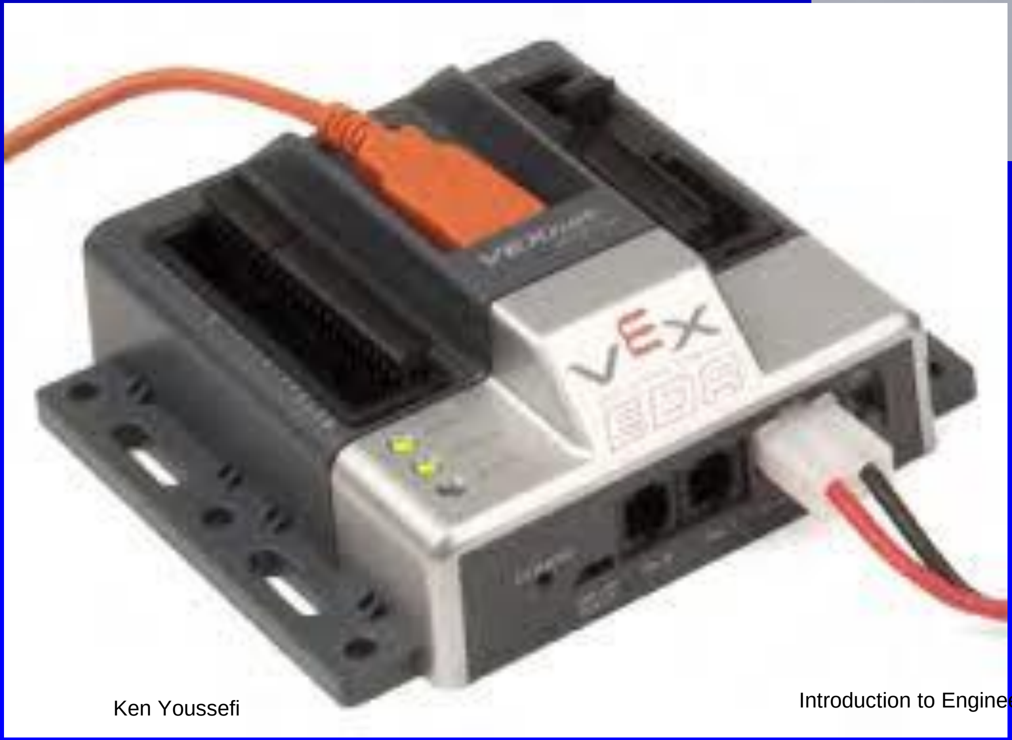
On/Off switch

Backup Batter Port (not available)

Batter Ports

Orange cable (USB connections at both ends) used to download programs from computer to the controller

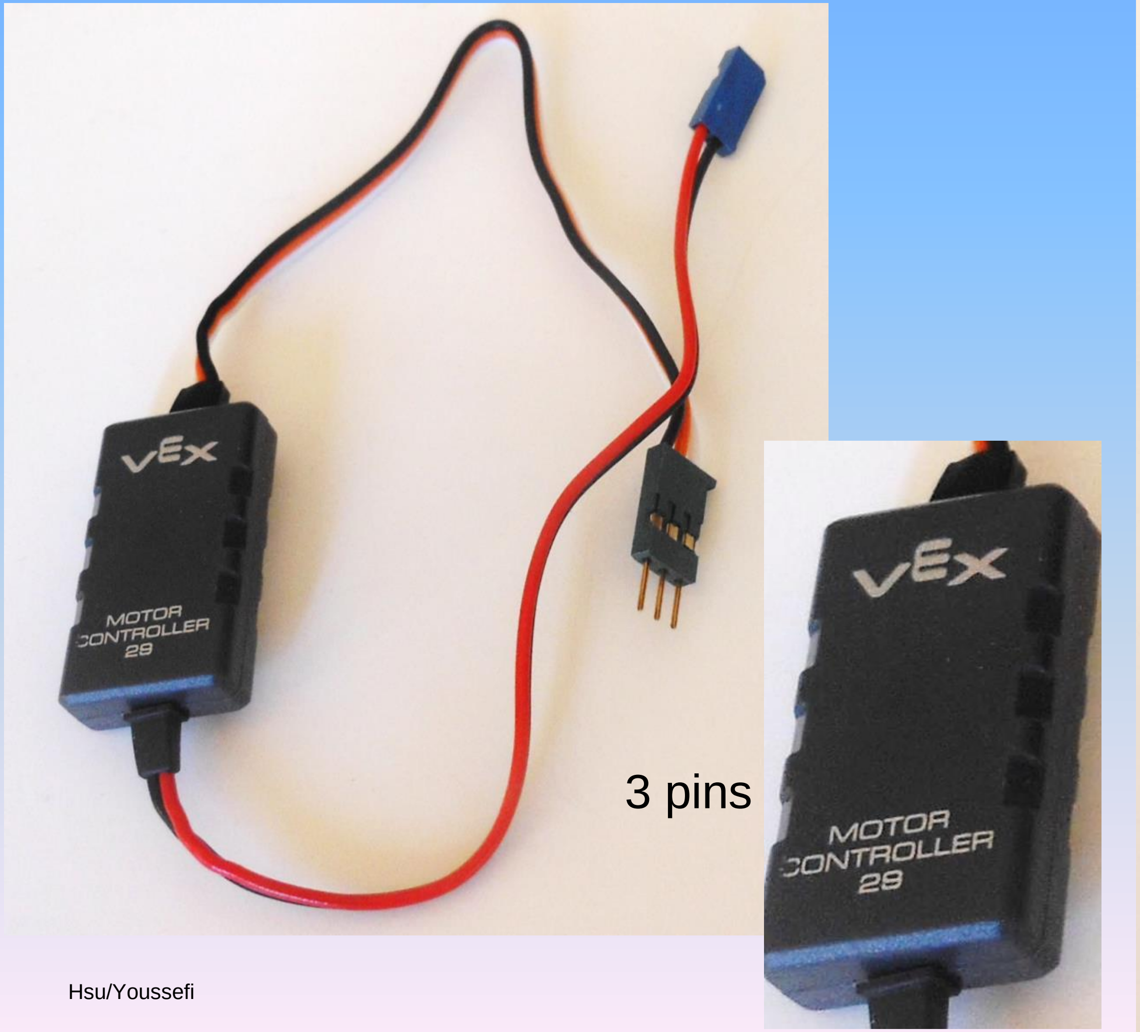


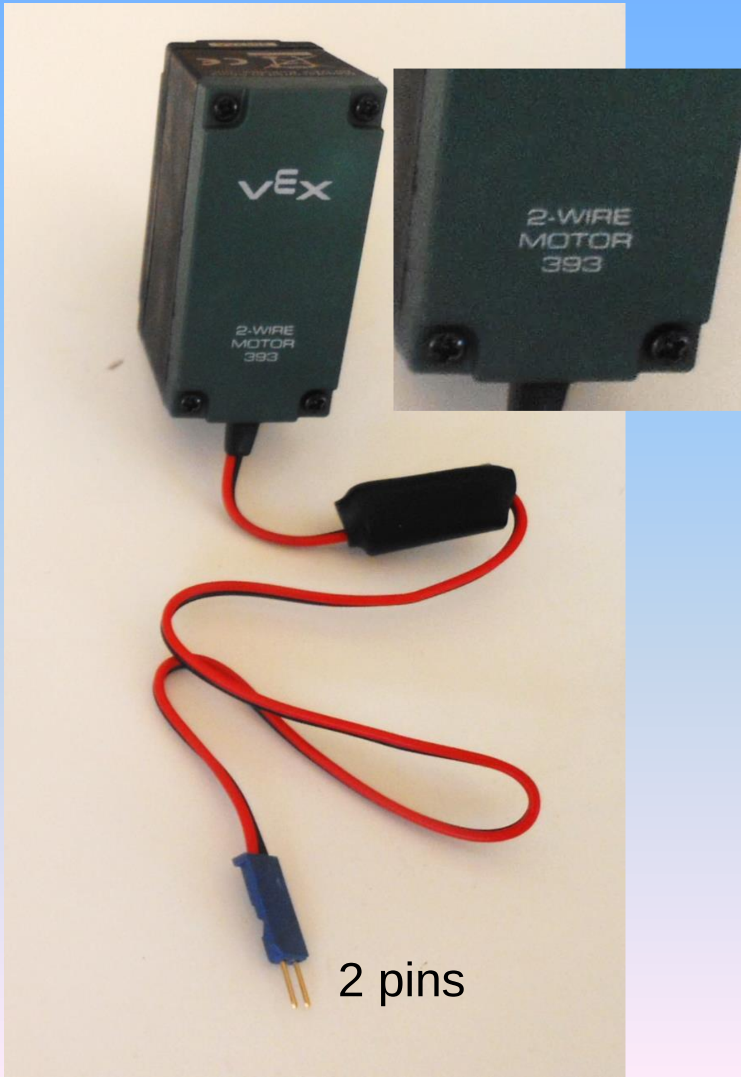


Battery connection

Mechanical Components – Motor (Cortex)

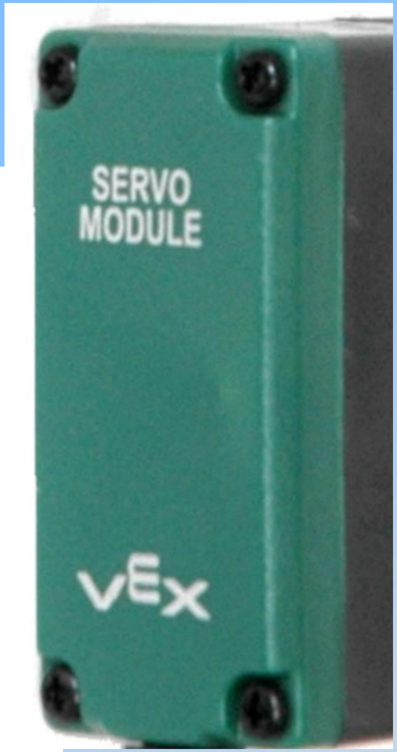
The new robot kit includes three 2-wire motors that can be converted to a 3-wire motor using the controller 29. Ports 1 and 10 on the new controller are designated for 2-wire motors only.

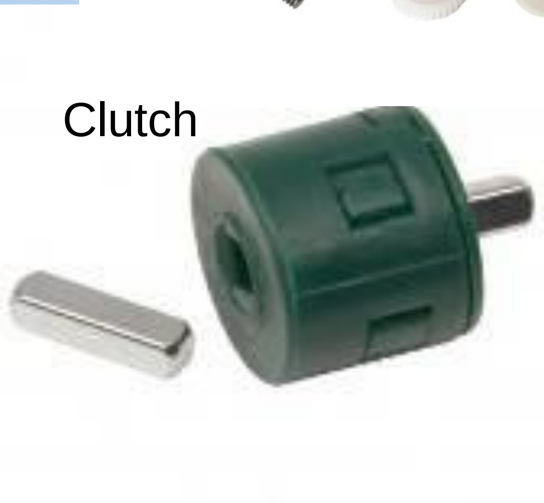




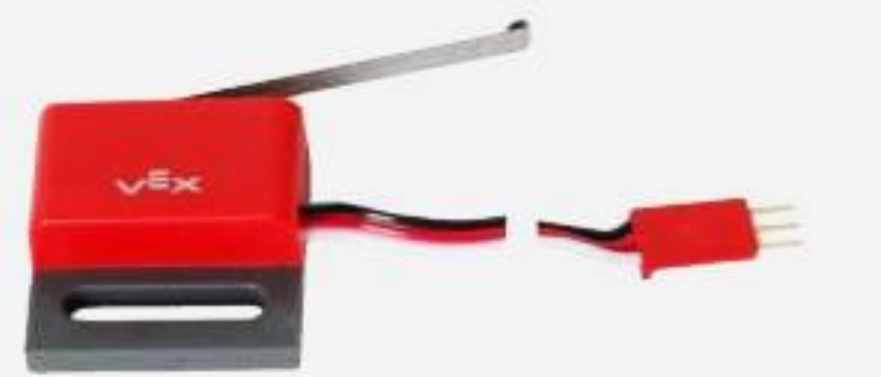
Servo and Clutch

Servo are used to control the rotation of a shaft (controls the motion of the robot arm). Make sure the clutch module is always used. It prevents damage to the motor in case of large torque.

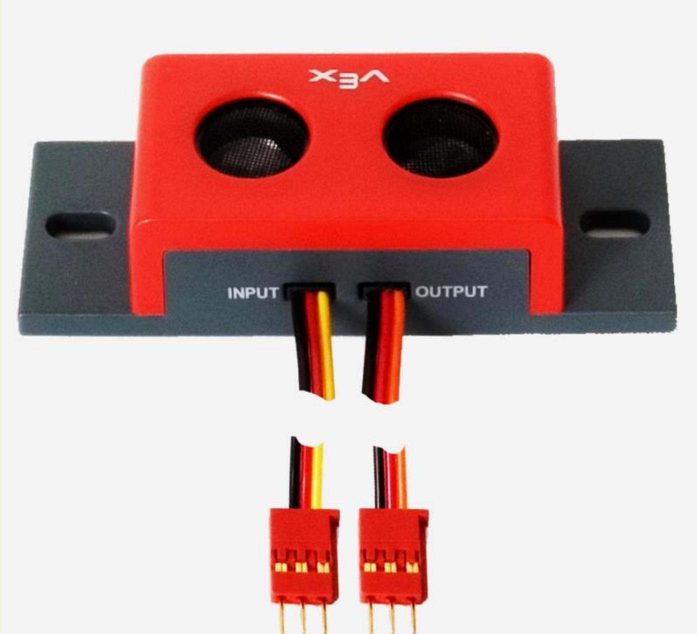



Clutch

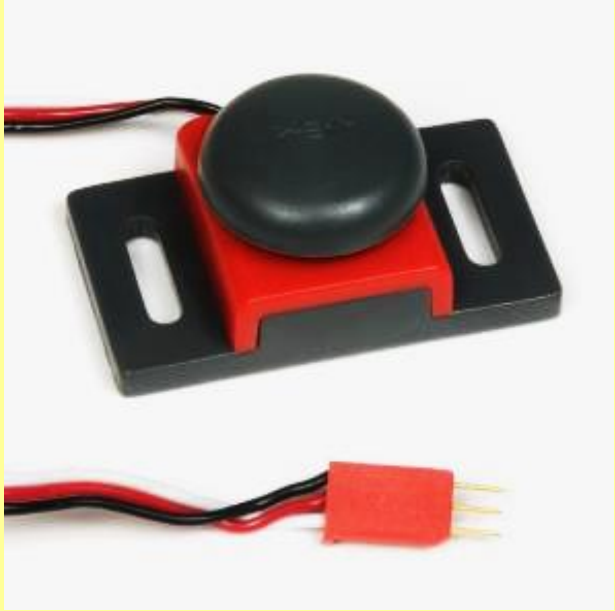
VEX Sensors



Limit Switch Sensor

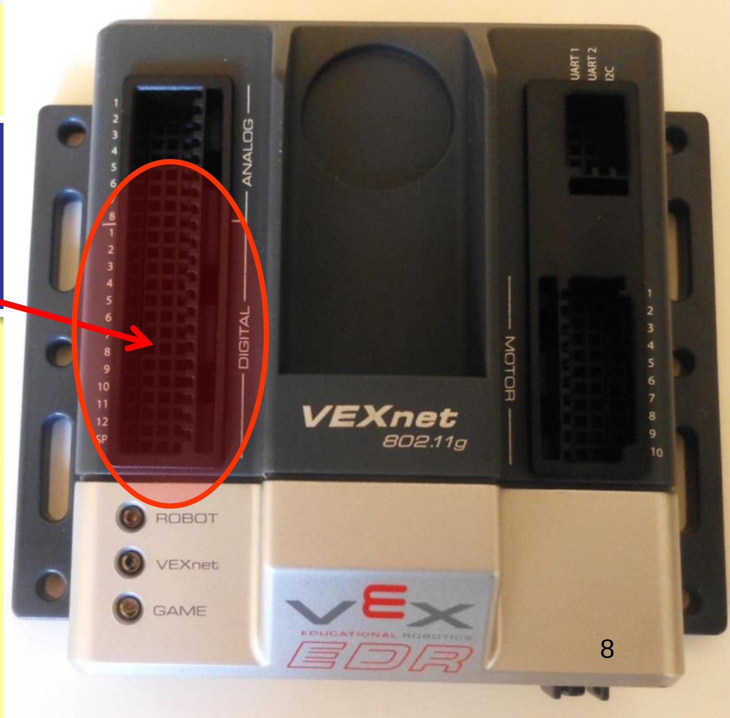


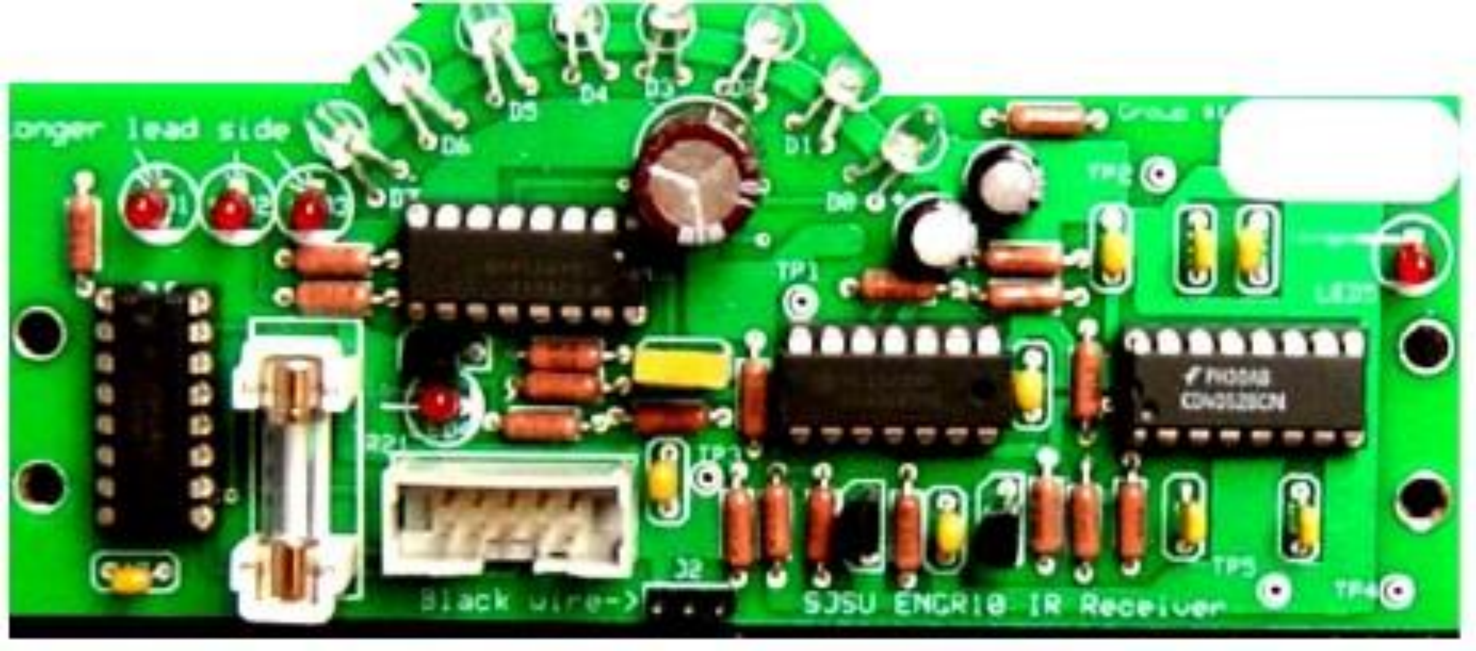
Ultrasonic Sensor



Bumper Switch Sensor

Attach all digital sensors to 1-12 ports in Digital section





8 Detectors, cover a field of view of 100^o

Infrared Receiver Board (IRB), the eyes of the robot.

Do's and Don't's

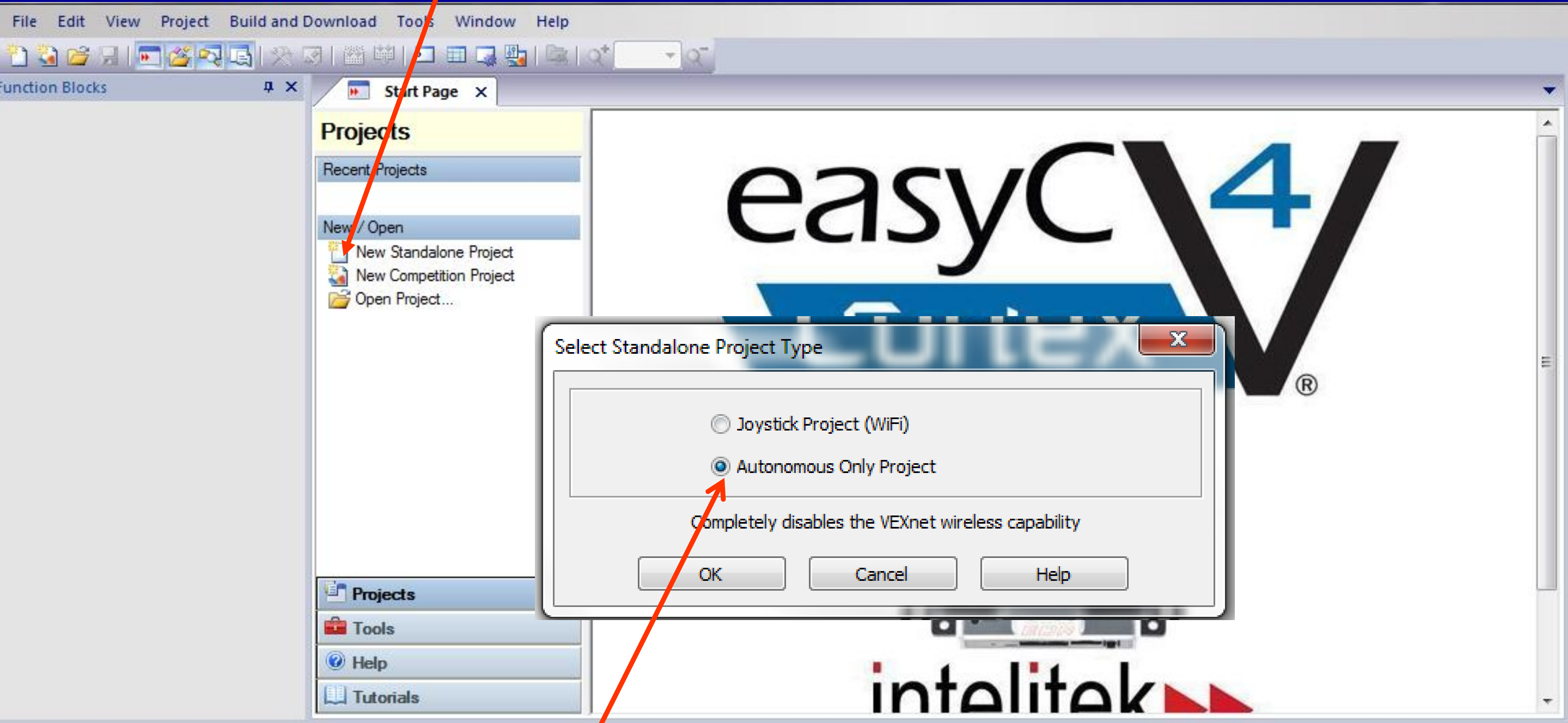
- Make sure the battery is charged.
- Do not place the VEX manual in your box, leave it on the table.
- Do not tamper with other group's robot. Do not leave your box on the table, store it on top of the work bench that includes your box number.
- Clean up before leaving
- There are two size of allen wrenches used for tightening screws. Leave the allen wrenches on the table for all to share.

Design Considerations

- The best design is the simplest design. Make your robot using the fewest number of parts.
- Plan ahead the location of the major components (controller, battery, electronic board, the arm, sensors The electronic board should be in the front and the white LEDs should have an unobstructed view of the beacon. The height of the LEDs should be roughly the same as the ones in the beacon.
- The electronic board should be ***insulated*** from the metal parts of the robot. Otherwise the IRB will be shorted
- You will have to plug and unplug components from the controller, leave space so the top of controller is readily accessible.

EasyCcortex

Select New Standalone Project



Select Standalone Project Type

○ Joystick Project (WiFi)

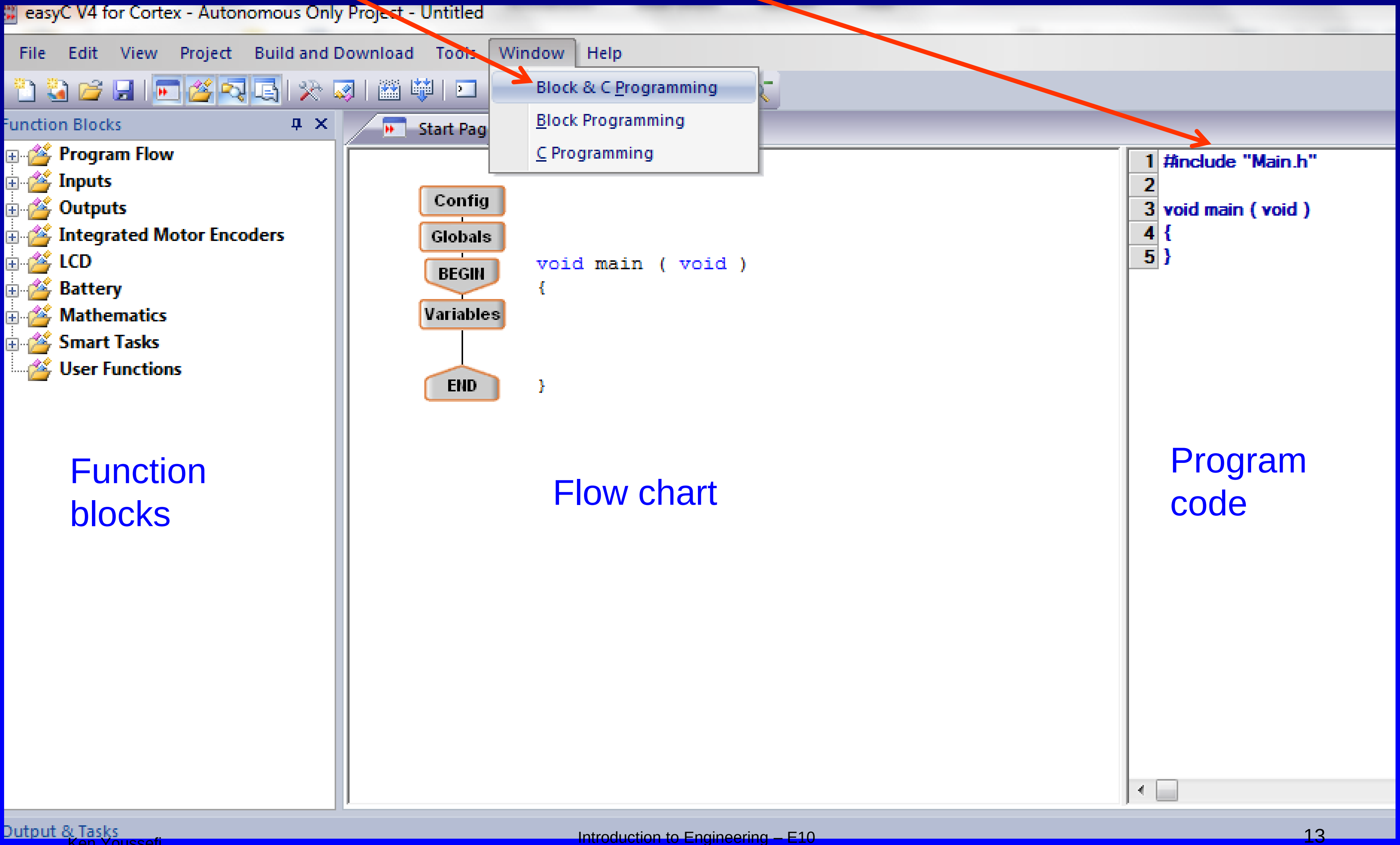
● Autonomous Only Project

Completely disables the VEXnet wireless capability

OK Cancel Help

Choose Autonomous Only Project

From Window option select the Block & C Programing option to view the code



```
void main ( void )
{


}
```

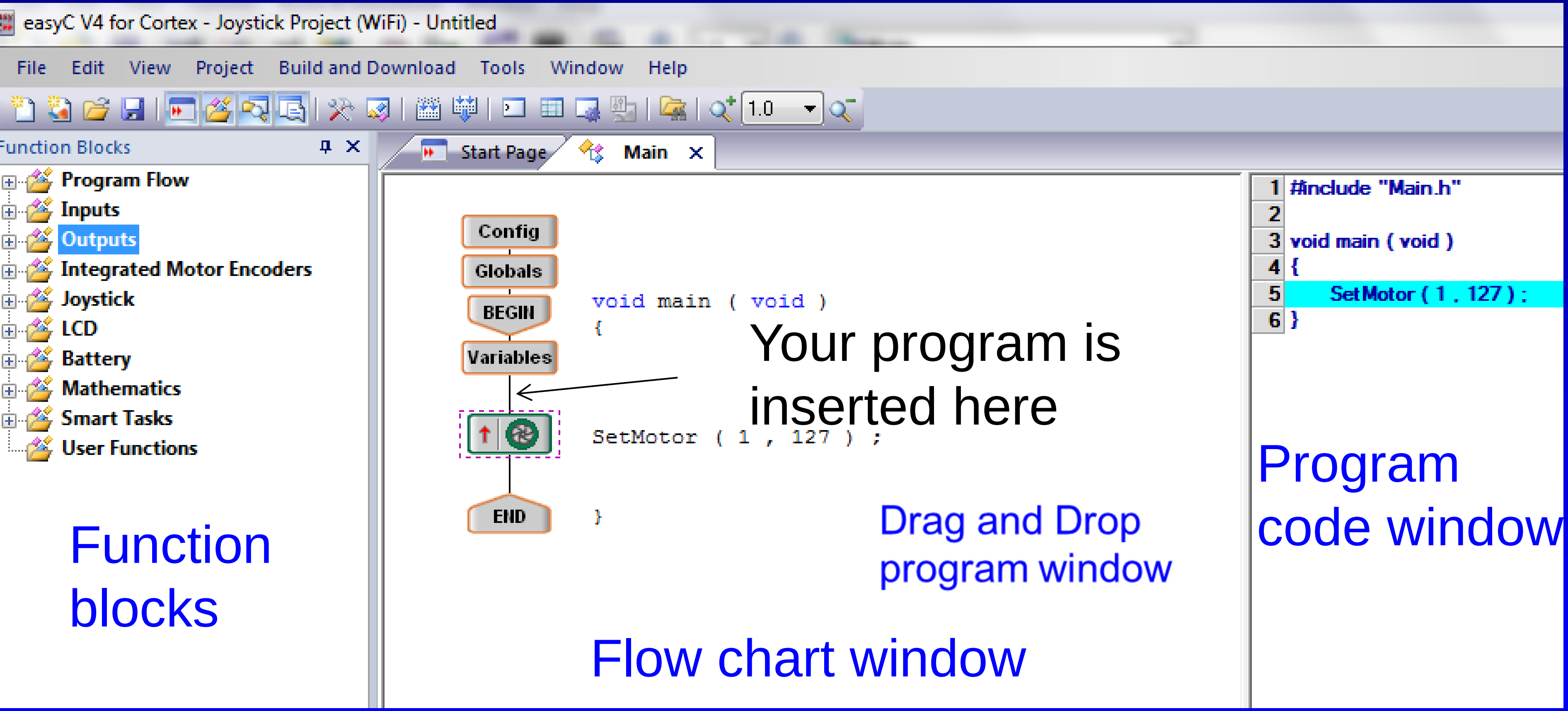
```
#include "Main.h"

void main ( void )
{
}
```

Function blocks

Flow chart

Program code

EasyCcortex Layout



```
void main ( void )
{
SetMotor ( 1 , 127 ) ;
}
```

```
#include "Main.h"

void main ( void )
{
    SetMotor ( 1 , 127 ) ;
}
```

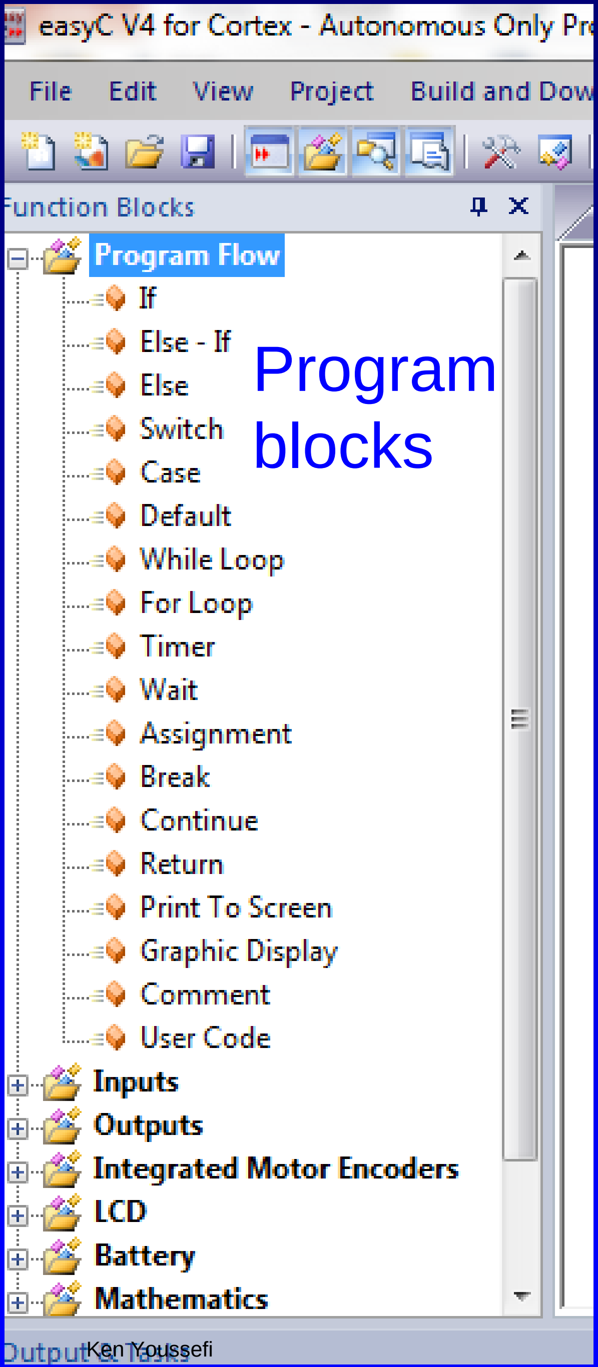
Your program is inserted here

Function blocks

Drag and Drop program window

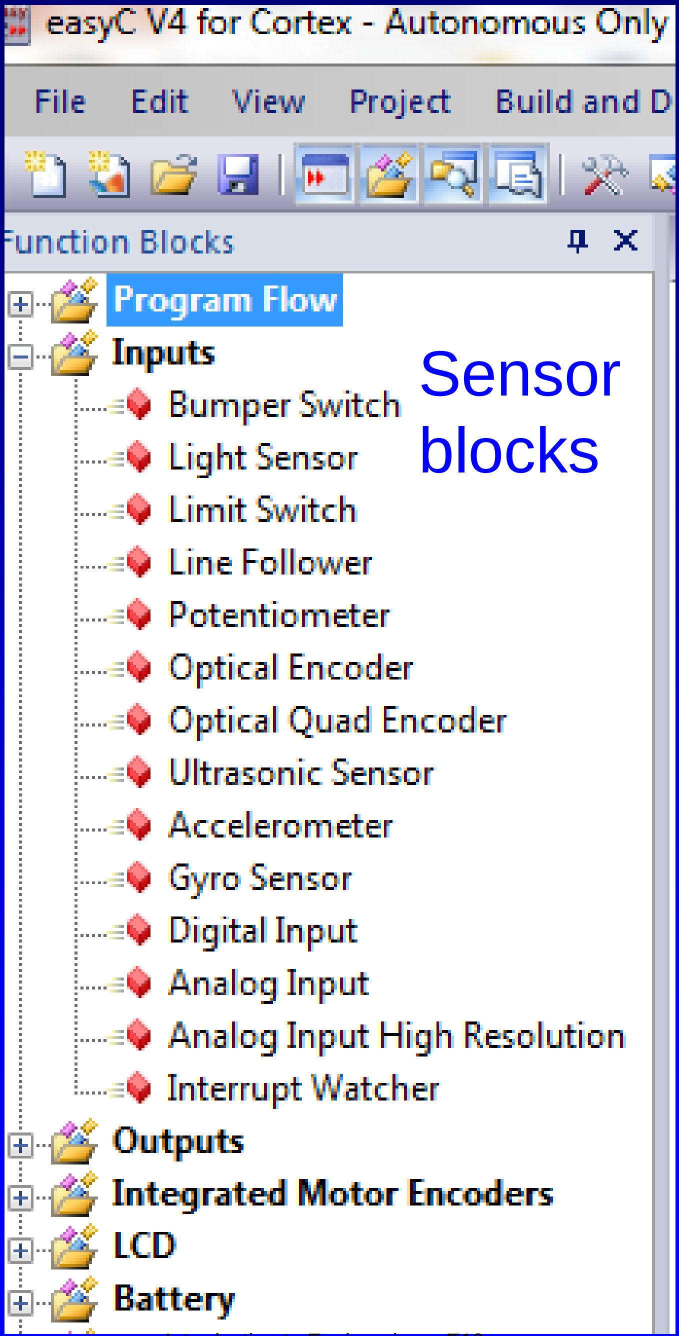
Flow chart window

Program code window



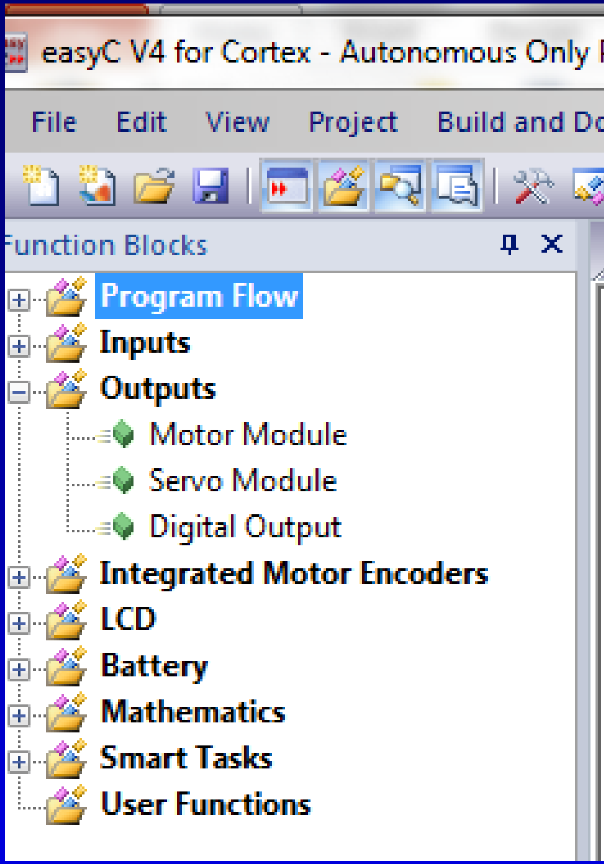
- Program Flow
  - If
  - Else - If
  - Else
  - Switch
  - Case
  - Default
  - While Loop
  - For Loop
  - Timer
  - Wait
  - Assignment
  - Break
  - Continue
  - Return
  - Print To Screen
  - Graphic Display
  - Comment
  - User Code
- Inputs
- Outputs
- Integrated Motor Encoders
- LCD
- Battery
- Mathematics

Program blocks



- Program Flow
- Inputs
  - Bumper Switch
  - Light Sensor
  - Limit Switch
  - Line Follower
  - Potentiometer
  - Optical Encoder
  - Optical Quad Encoder
  - Ultrasonic Sensor
  - Accelerometer
  - Gyro Sensor
  - Digital Input
  - Analog Input
  - Analog Input High Resolution
  - Interrupt Watcher
- Outputs
- Integrated Motor Encoders
- LCD
- Battery

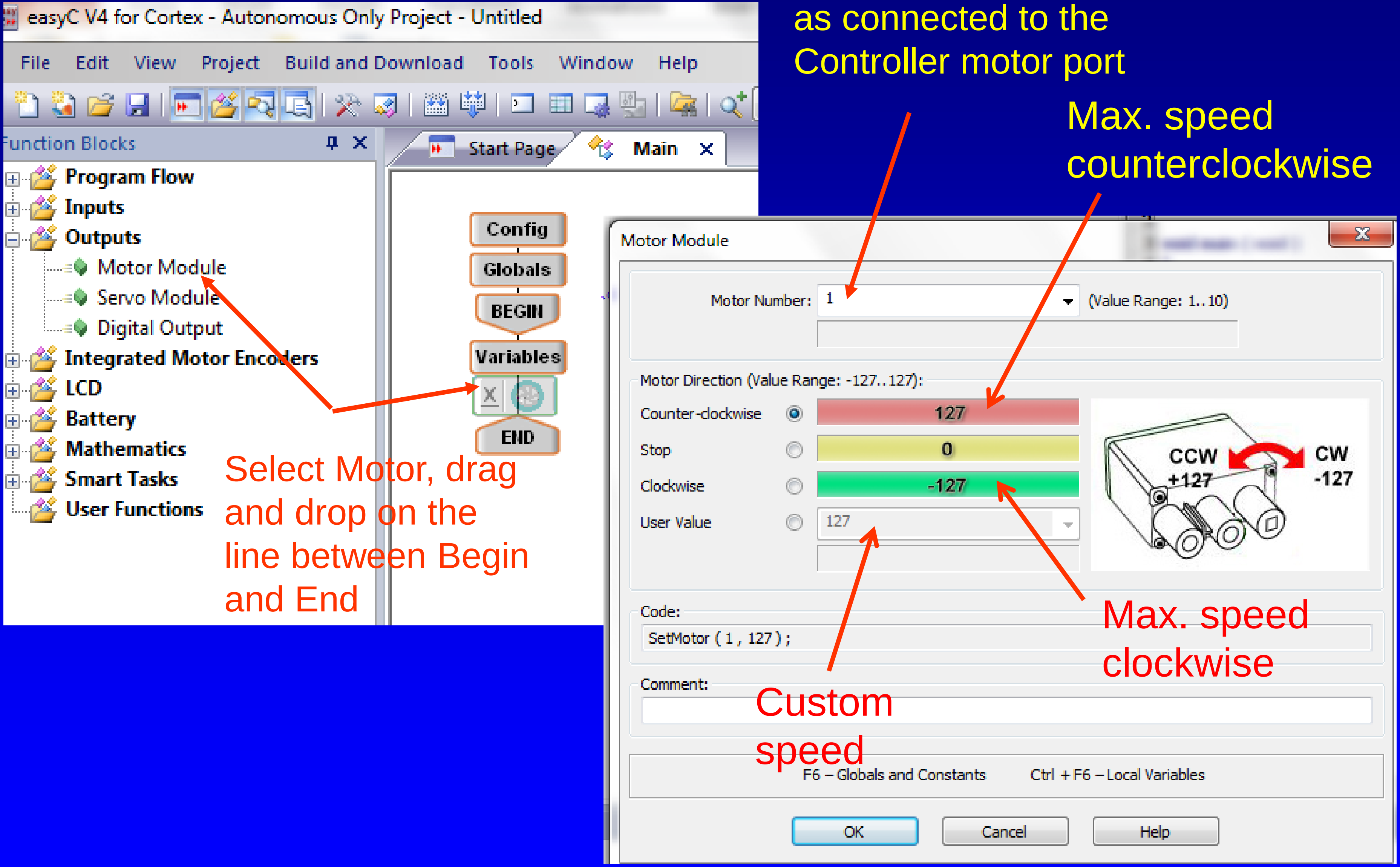
Sensor blocks



- Program Flow
- Inputs
- Outputs
  - Motor Module
  - Servo Module
  - Digital Output
- Integrated Motor Encoders
- LCD
- Battery
- Mathematics
- Smart Tasks
- User Functions

Outputs options

Adding Continuous motor



Motor Number: 1 (Value Range: 1..10)

Motor Direction (Value Range: -127..127):

Counter-clockwise	●	127
Stop	○	0
Clockwise	○	-127
User Value	○	127

Code:

```
SetMotor ( 1 , 127 ) ;
```

Comment:

F6 – Globals and Constants   Ctrl + F6 – Local Variables

Enter motor number, as connected to the Controller motor port

Max. speed counterclockwise

Select Motor, drag and drop on the line between Begin and End

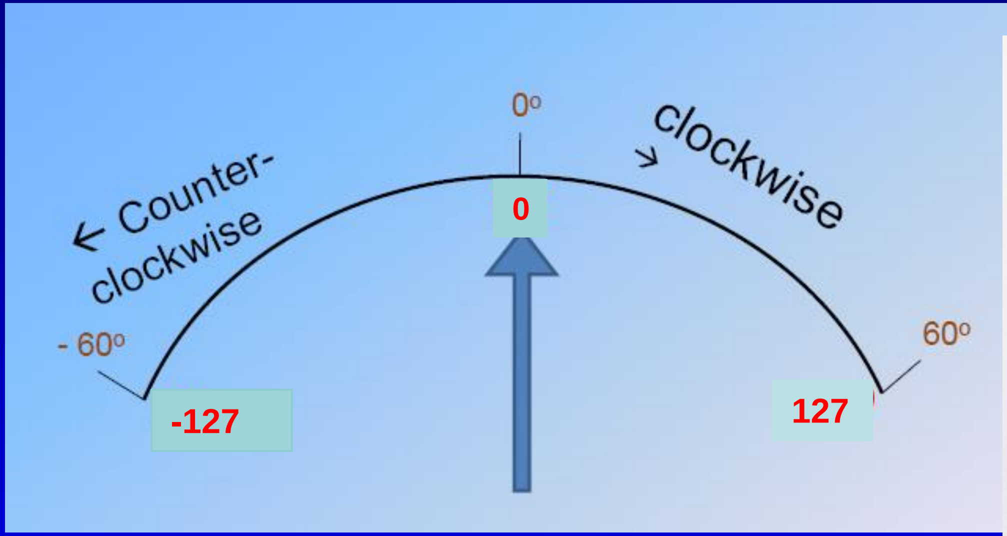
Max. speed clockwise

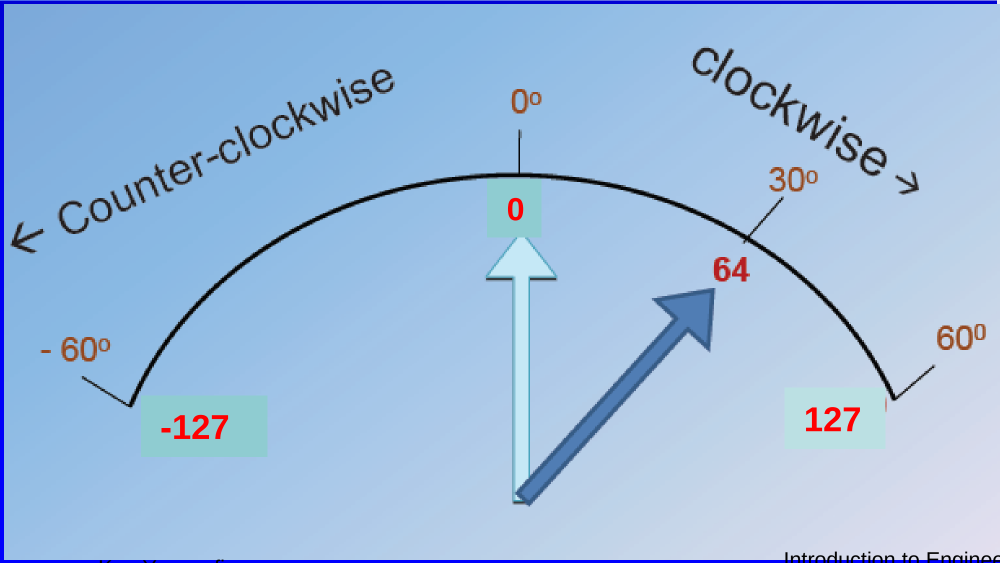
Custom speed

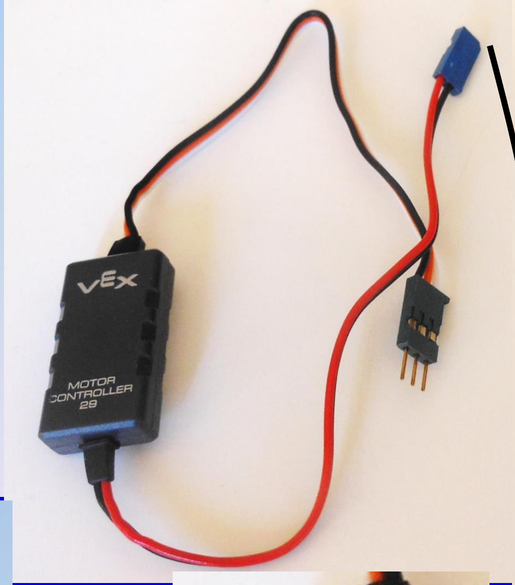
Servo motor

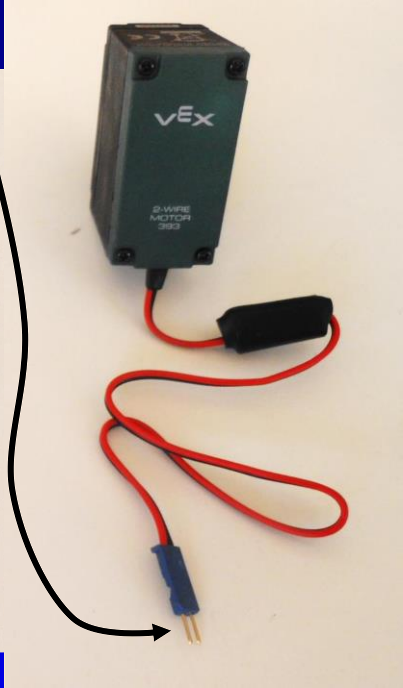
Servos control the position of the motor shaft, angle of rotation

Connect the Motor Controller 29 to the 2-wire Motor. This converts a 2-wire motor to a 3-wire servo. **Connect Red to Red wire**

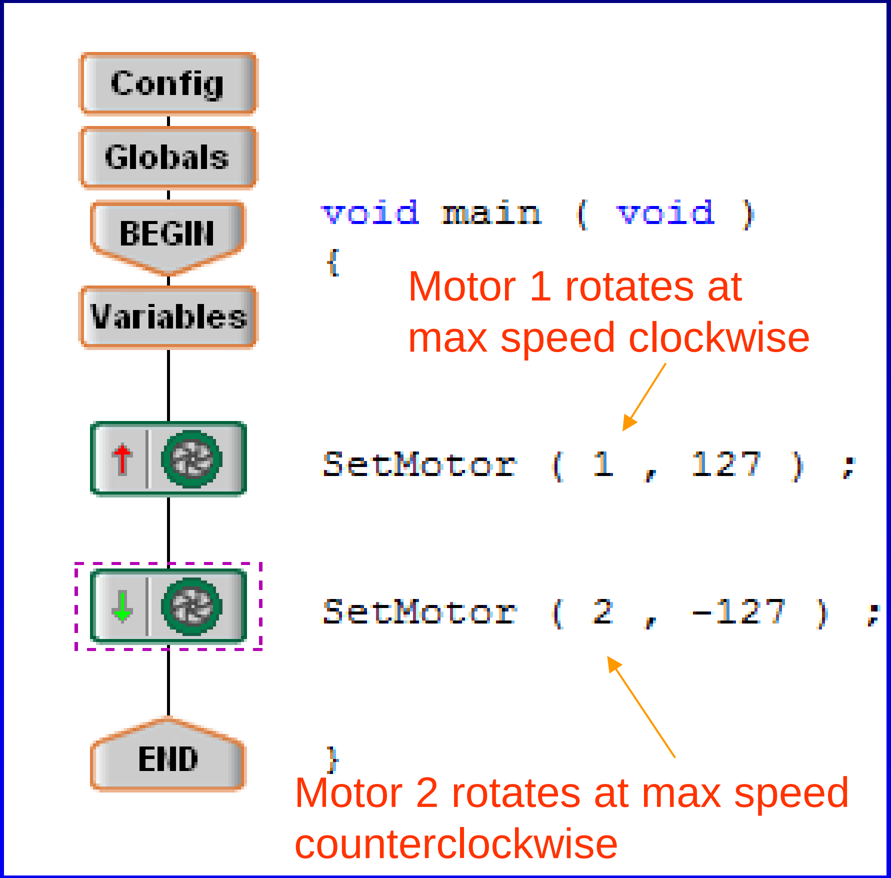








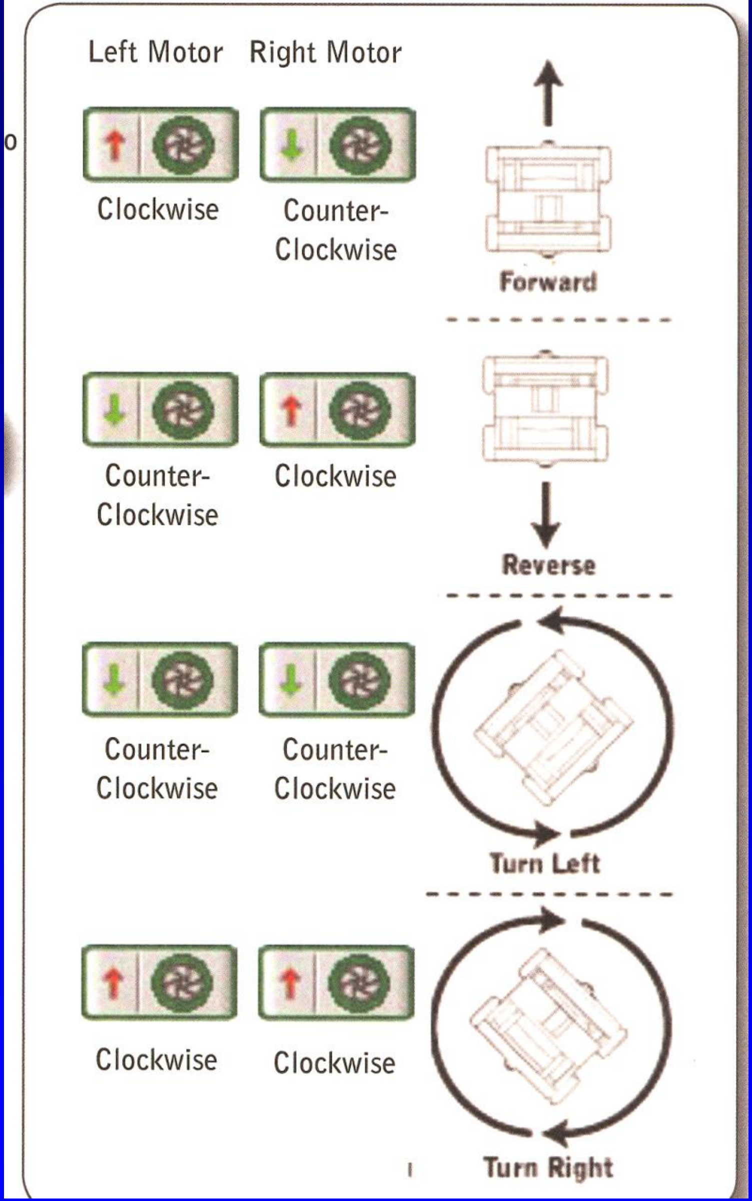
Robot moving forward



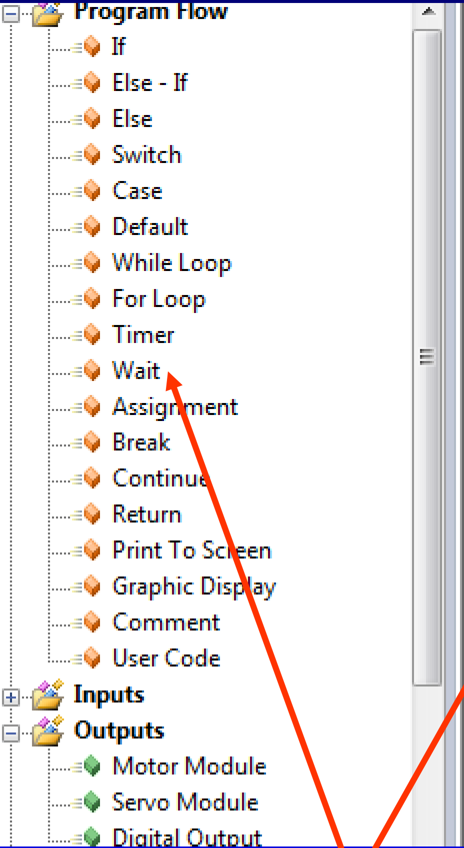
```
void main ( void )
{
        SetMotor ( 1 , 127 ) ;
        SetMotor ( 2 , -127 ) ;
}
```

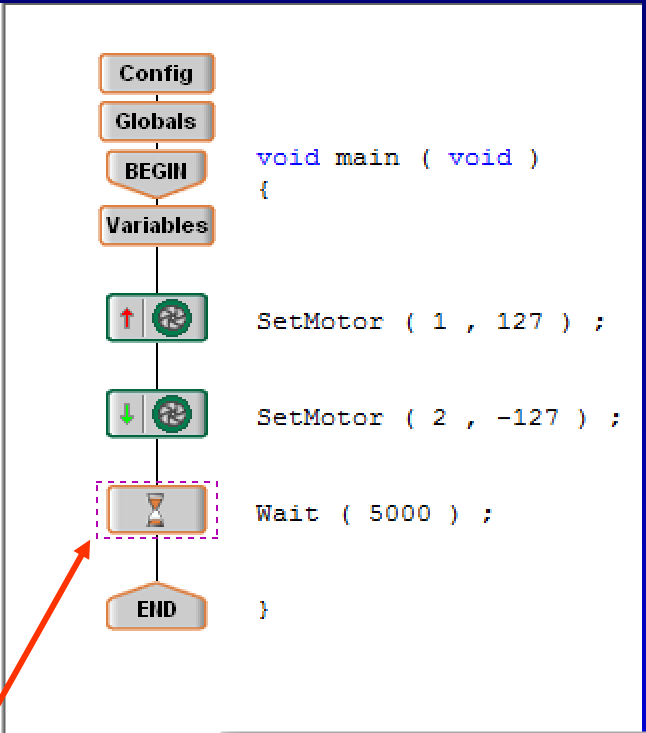
Motor 1 rotates at max speed clockwise

Motor 2 rotates at max speed counterclockwise



Robot moves forward at high speed

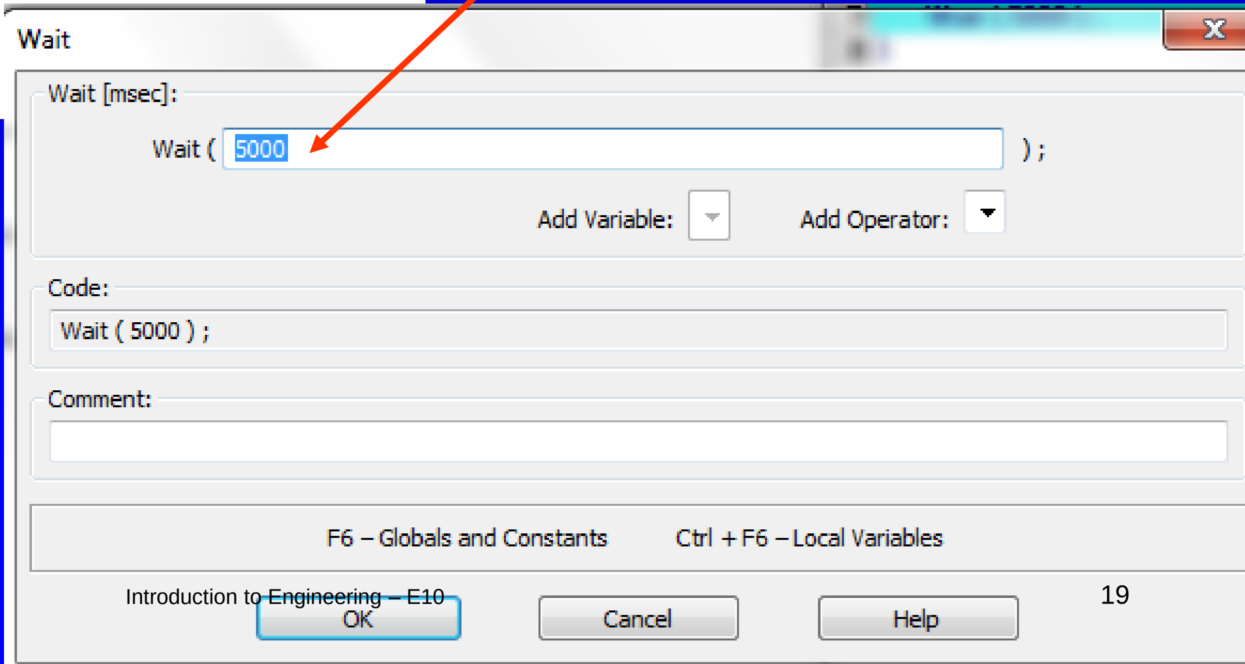




```
void main ( void )
{
SetMotor ( 1 , 127 ) ;
SetMotor ( 2 , -127 ) ;
Wait ( 5000 ) ;
}
```

The Wait function

Enter time in millisecond (5000 msec)



Wait [msec]:

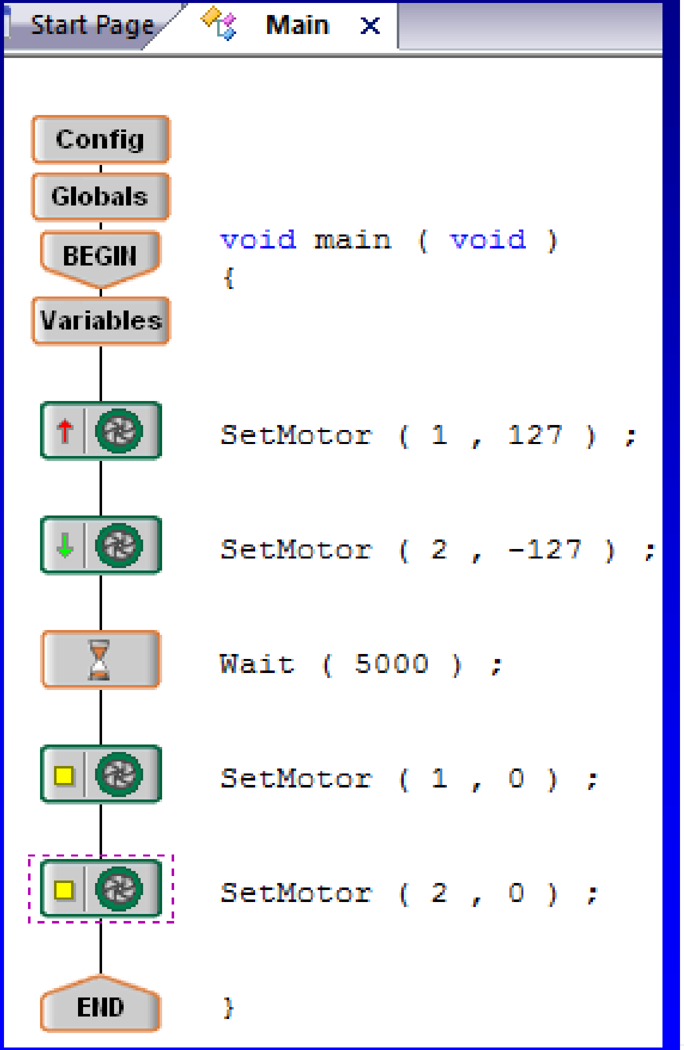
Wait (5000) ;

Code:

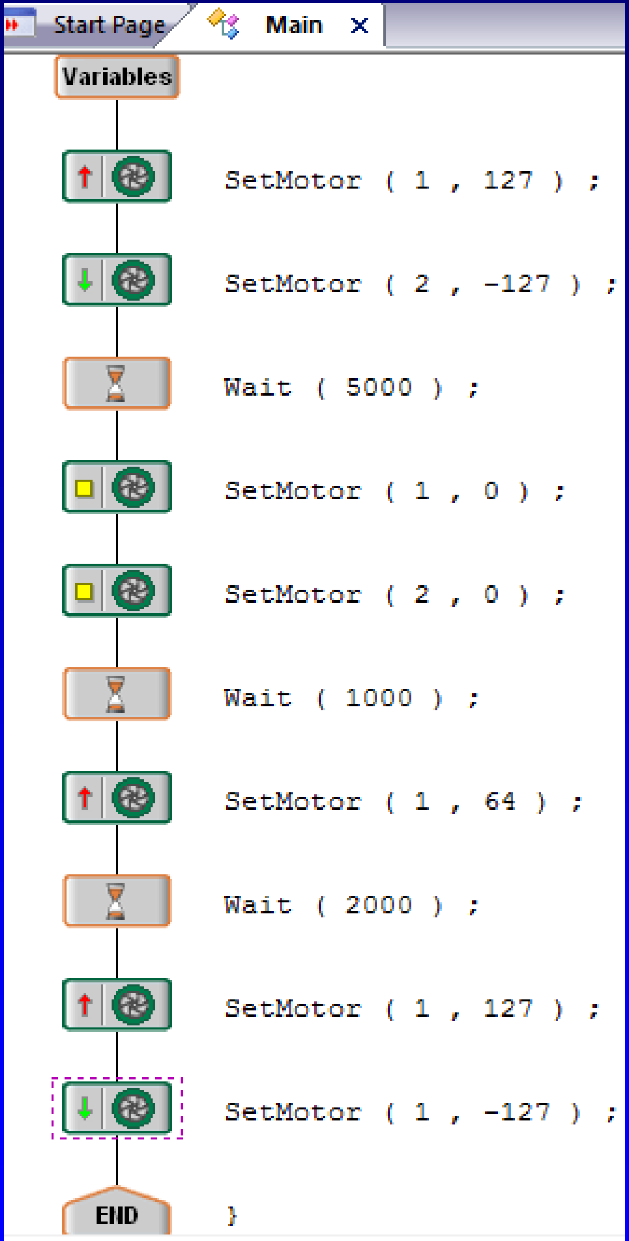
Wait (5000) ;

Comment:

Robot goes forward at top speed for 5 seconds and then stops



```
void main ( void )
{
SetMotor ( 1 , 127 ) ;
SetMotor ( 2 , -127 ) ;
Wait ( 5000 ) ;
SetMotor ( 1 , 0 ) ;
SetMotor ( 2 , 0 ) ;
}
```



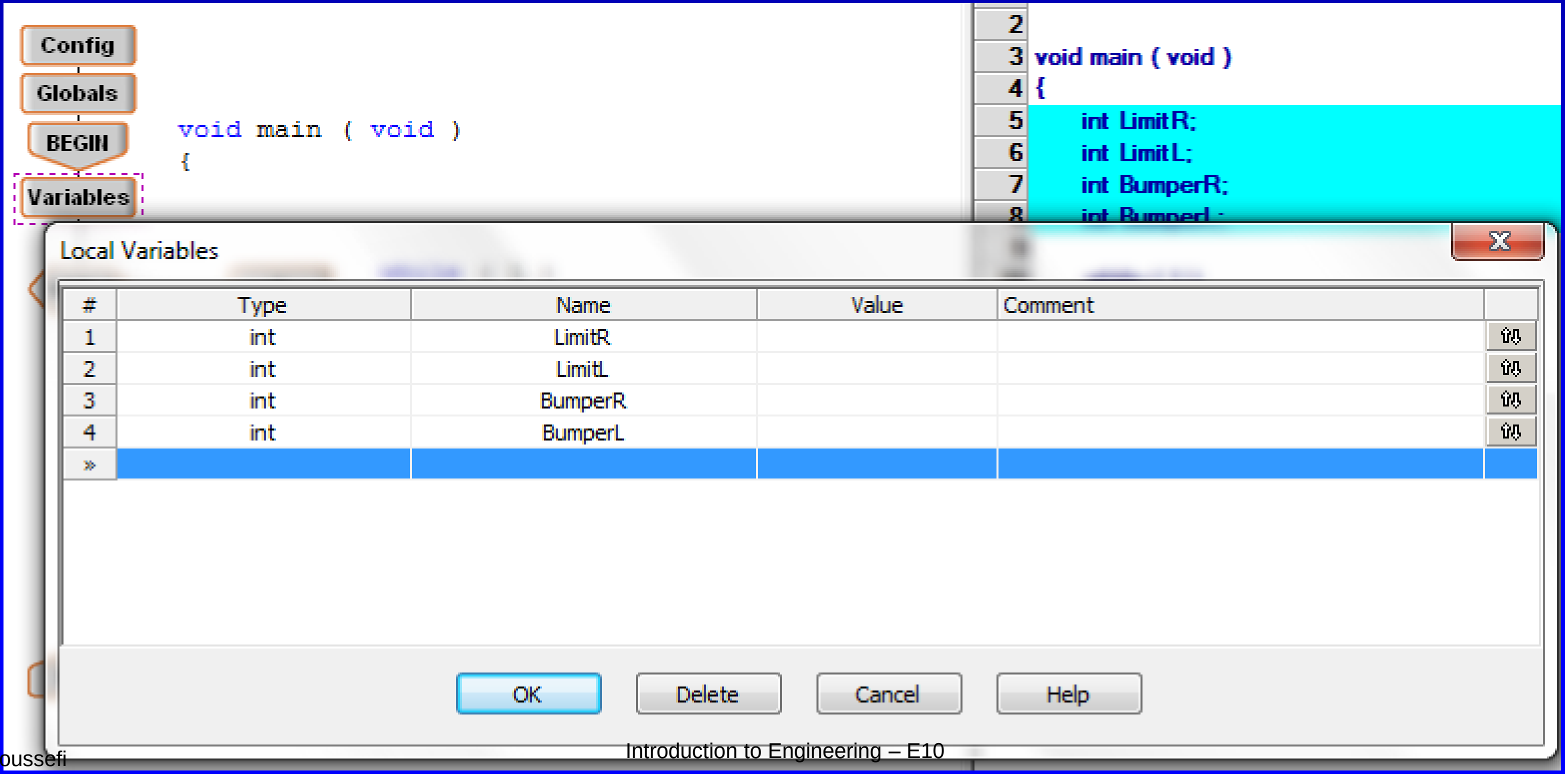
```
SetMotor ( 1 , 127 ) ;
SetMotor ( 2 , -127 ) ;
Wait ( 5000 ) ;
SetMotor ( 1 , 0 ) ;
SetMotor ( 2 , 0 ) ;
Wait ( 1000 ) ;
SetMotor ( 1 , 64 ) ;
Wait ( 2000 ) ;
SetMotor ( 1 , 127 ) ;
SetMotor ( 1 , -127 ) ;
}
```

Robot goes forward at top speed for 5 seconds and then stops. It waits 1 second and then it turns right at half speed for 2 seconds and goes forward.

Objective

Write a program so that the robot, upon encountering any obstacle, would back up, turn and continue.

Define all sensors: Limit and Bumper switches on the right and left side of the robot. Right click on the Variable and Edit the Block



```
void main ( void )
{
```

```
void main ( void )
{
    int LimitR;
    int LimitL;
    int BumperR;
    int BumperL;
```

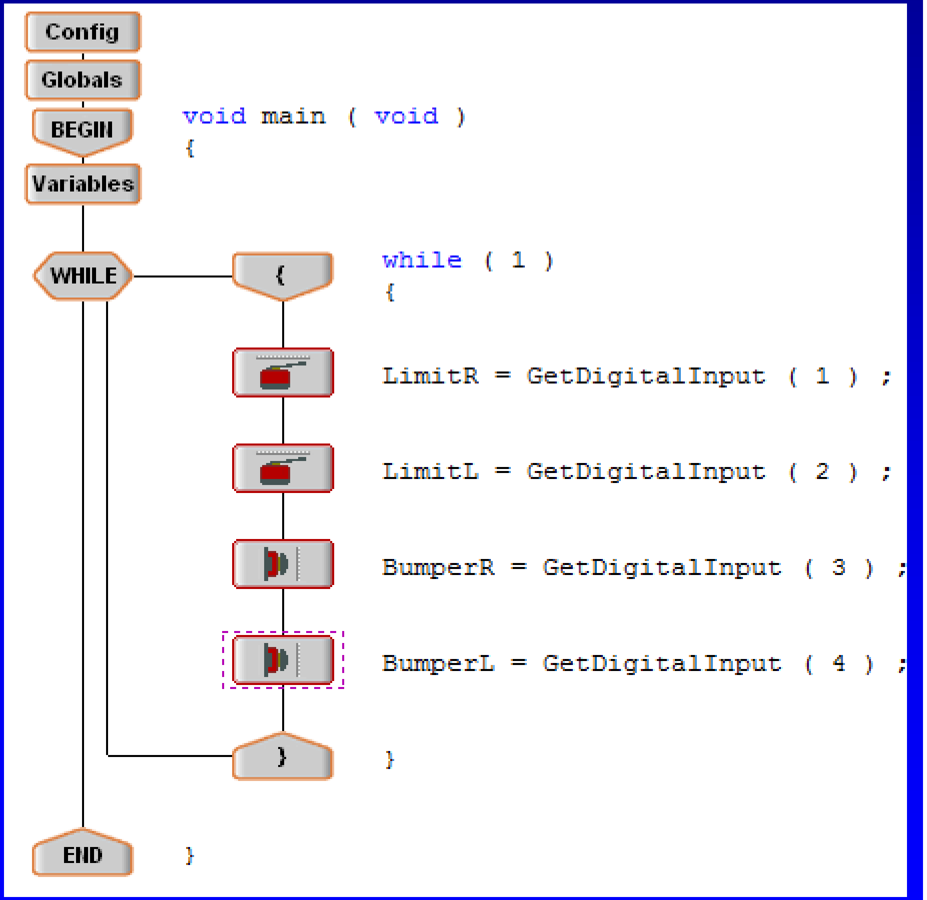
Local Variables

#	Type	Name	Value	Comment
1	int	LimitR		
2	int	LimitL		
3	int	BumperR		
4	int	BumperL		
»				

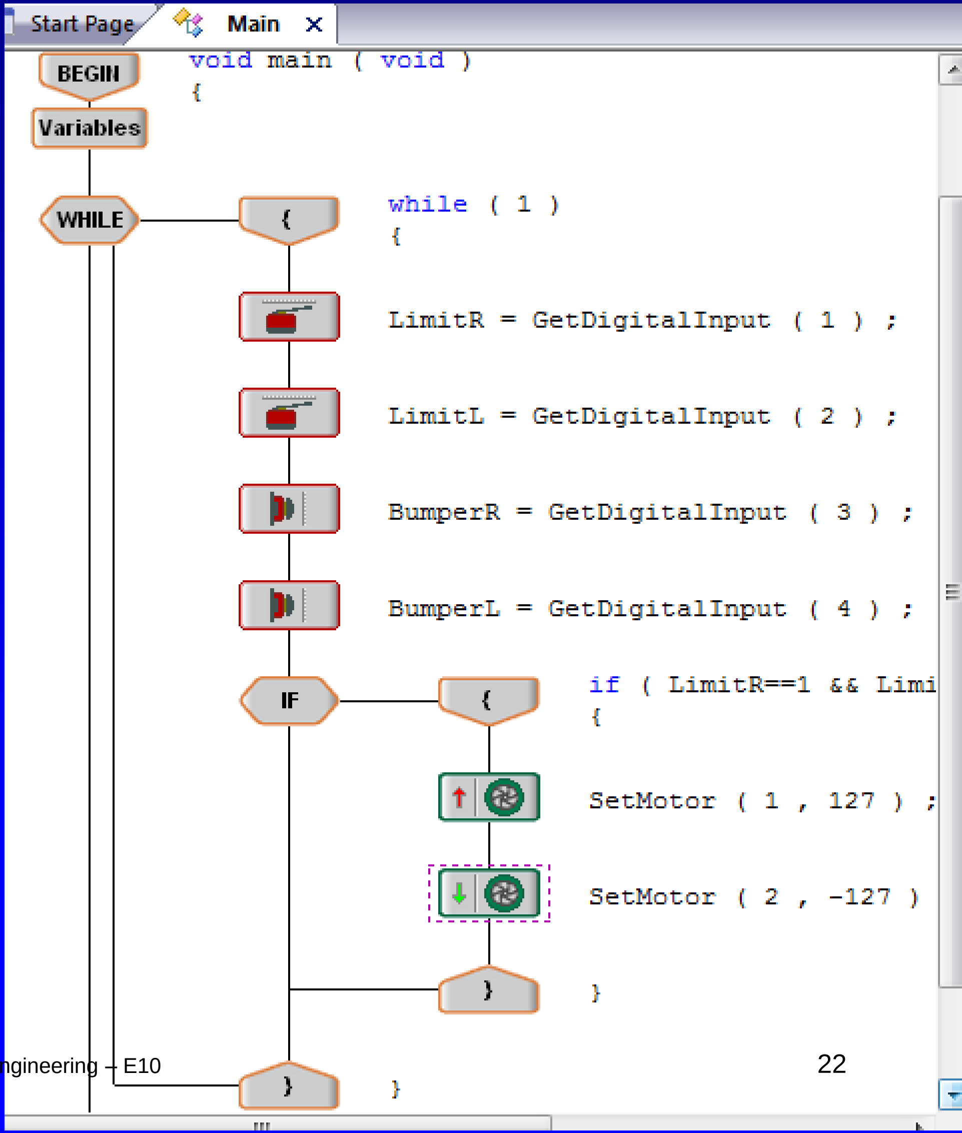
Inside a While Loop, drag and drop all sensors

Add an ***IF*** statement to check if the sensors have been activated or not (robot has hit an obstacle or not)

Initializing and scanning the sensors

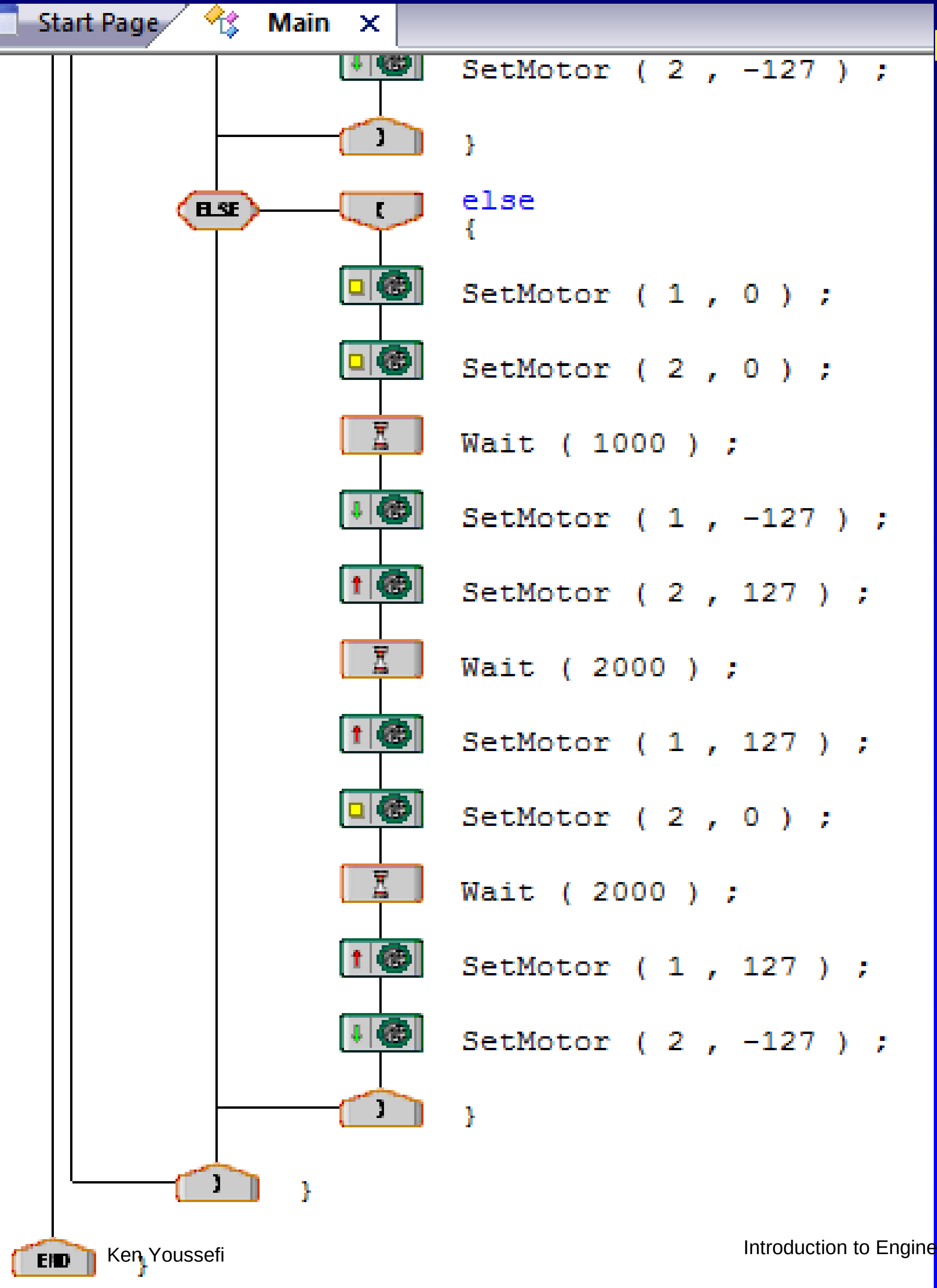


```
void main ( void )
{
while ( 1 )
{
LimitR = GetDigitalInput ( 1 ) ;
LimitL = GetDigitalInput ( 2 ) ;
BumperR = GetDigitalInput ( 3 ) ;
BumperL = GetDigitalInput ( 4 ) ;
}
}
```



```
void main ( void )
{
while ( 1 )
{
LimitR = GetDigitalInput ( 1 ) ;
LimitL = GetDigitalInput ( 2 ) ;
BumperR = GetDigitalInput ( 3 ) ;
BumperL = GetDigitalInput ( 4 ) ;
if ( LimitR==1 && Limi
{
SetMotor ( 1 , 127 ) ;
SetMotor ( 2 , -127 )
}
}
```

ELSE → one of the sensors is activated (o)



```
SetMotor ( 2 , -127 ) ;
}
else
{
SetMotor ( 1 , 0 ) ;
SetMotor ( 2 , 0 ) ;
Wait ( 1000 ) ;
SetMotor ( 1 , -127 ) ;
SetMotor ( 2 , 127 ) ;
Wait ( 2000 ) ;
SetMotor ( 1 , 127 ) ;
SetMotor ( 2 , 0 ) ;
Wait ( 2000 ) ;
SetMotor ( 1 , 127 ) ;
SetMotor ( 2 , -127 ) ;
}
}
}
```

Stop both motors for 1 second

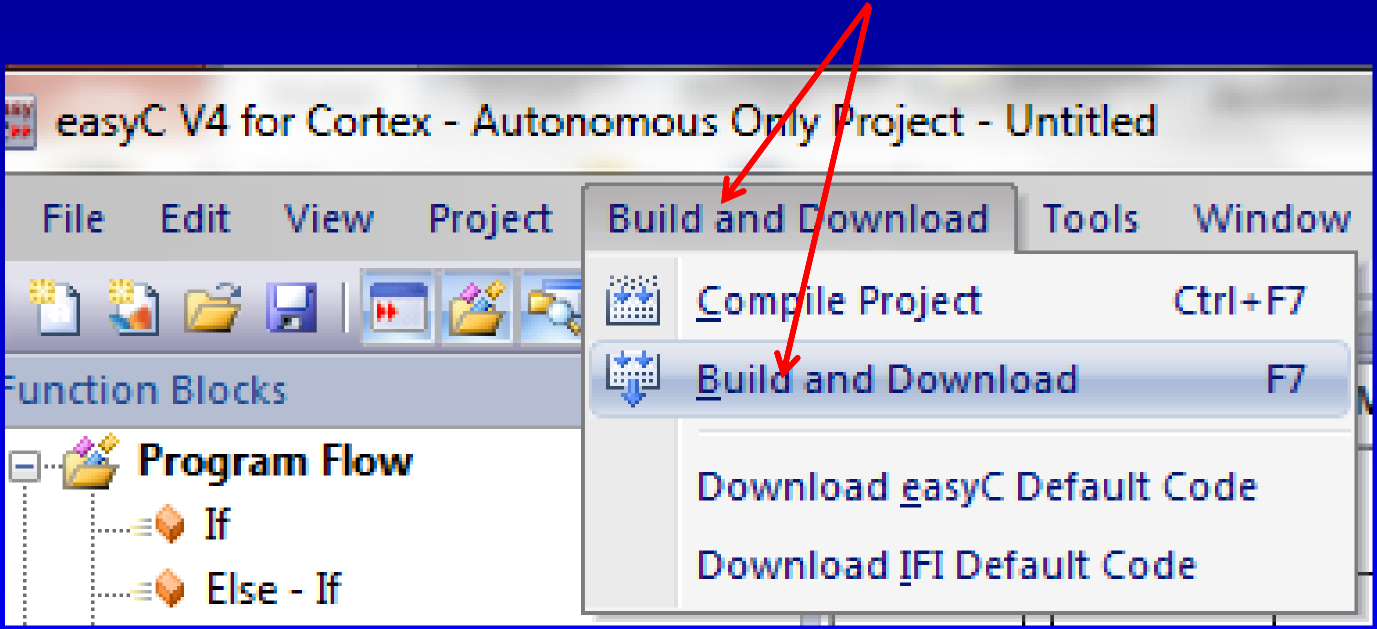
Back up for 2 seconds

Turn for 2 seconds

Go forward

Downloading the Program to the Controller

Connect the PC to the Controller using the USB orange cord. Select ***Build and Download*** option.



Follow the on screen instructions. Check for compilation errors in the Program Code section.