Programming the VEX Robot

Preparing for Programming

Setup

Before we can begin programming, we have to set up the computer we are using and the robot/controller. We should already have:

- Windows (XP or later) system with easy-C installed
- VEX Robot and joystick
- “A” to “A” (male to male) USB connection cable

Loading Drivers

We need to load the PL-2303 controller driver (if not already done). We do this by running the VEX_USB2 software installed as part of the Easy-C install (or available on the Easy-C CD)

Upgrading the Firmware

Firmware is a “code” layer between the hardware and the user software. It is often considered part of the hardware in that it is always available (persistent) but it is special software that runs on the hardware to provide an interface between the hardware and the user-developed software. On your computer, this is the BIOS that you see starting before Window. Vendors occasionally provide updates to their firmware; we need to load these updates onto the devices.

The process for upgrading the robot:

1) Power off the robot
2) Remove the VEXnet device from the robot
3) Plug the USB cable into the computer
4) Press the “config” button on the microcontroller. Plug the USB cable to the microcontroller. Wait for 3 green lights to come on, and then release the “config” button. You should see the “Game” light flashing quickly and the “Robot” light slowly blinking. You may see messages on your Windows machine about a new device being installed.
5) Start the IFI VEXnet Firmware Upgrade utility (Start/All Programs/EasyC v4 for Cortext/iFIVEXnet Firmware Upgrade).
    a. Click the “Search” button. Assuming everything is set up correctly, the Search button will change and become disabled, and the “Download” button will be enabled.
    b. Click the “Download” button. The firmware upgrade will begin. A progress bar will show the progress of the upgrade. The controller lights should be that VEXnet is blinking red and Robot is blinking green during the upgrade.
    c. When done, the dialog should indicate that the Download is complete, and that VEXnet is up-to-date and no further action is necessary. Close this dialog, and unplug the USB cable from the robot.
6) Power off the joystick, and remove the VEXnet USB device

7) Press the "config" button on the joystick and insert the USB cable connection. The joystick light should be blinking red while the VEXnet light is green. You may hear the "new device" sound from your computer.
8) Start the IFI VEXnet Firmware Upgrade Utility
    a. Click the "Search" button. Wait until the "Download" button becomes enabled.
    b. Click the "Download" button. Wait for the firmware upgrade to complete.
    c. Close the utility, and unplug the USB cable from the joystick. Do not forget to put the VEXnet device back in to the joystick!

Programming Introduction

Easy-C Introduction

Easy-C is a "code generator" in that we use it to generate C code. Other tools allow us to create C code directly; but it is easiest to use Easy-C as intended; as a way to generate code. Programmers do not need to know C to use it; however, being familiar with C will help, since the end result is C code.

Easy-C uses a "flowchart" approach to creating code. Objects (either programming constructs or "robot" items) are dragged into the appropriate place.

From Easy-C to the Robot

The process by which we get our code to the robot is:

1) Turn off the robot and remove the VEXnet device
2) Plug the USB connector into the PC and the robot
3) Using Easy-C, write your program
4) Using Easy-C, get your "program" onto the robot by using the menu options "Build and Download" and then "Build and Download". This will do the following:
    a. Easy-C will generate C code
    b. The C code will be compiled to "object" code
    c. A linker will combine your object code and libraries to create a "load module"
    d. The "load module" is then transferred via the USB cable to the robot
5) Remove the USB cable from the robot
6) Turn on the robot and it will run your code

Source Control

When you are entering code, you will want to periodically save the project to disk (in case the power goes out, there is a connection issue, etc.). However, when you save the project, you will be overwriting the existing one.

You may want to develop a naming convention for your projects that allow for incremental changes in "versions"; for example, you might decide to just add 1 to a number at the end of the project every time you make a substantial change (what does that mean? Big enough that you might want to be able to "go back" to the previous version). So the first save would be "project001"; the second, "project002", etc. You might want to use dates, or even incorporate which robot (if you have more than one) or which team (if you have more than one) into the name; like "Team X Robot A Score Ten Points Version 5".

Project Types

EasyC allows for the creation of two kinds of projects: StandAlone and Competition. The difference between the two is that the Competition projects have a shell that allows the game board to start autonomous mode and the stop it. We will not go over competition projects because it would be difficult to test them out; however, the programming for a StandAlone project is identical (in terms of what you write) as a Competition project.

StandAlone projects have two types: Joystick or Autonomous. If you select to create an Autonomous project, the VEXnet interface will not work when your code is running. This is very helpful in the testing process because you can run the code without anything plugged in to the USB port. However, if you want to be able to interact with the joystick, you need to create that type of project.

Reloading the "Default" Code

If you download code to the robot and it end up putting the robot in a "bad' state, you have to load a program to get it back to a "good" state. You can (and normally would) do that by fixing the problem with your code and then reloading it. However, if you want to get back to a know state, you can also load a "default" program provided in easyC. You do this by selecting "Download Default Code" from the "Build and Download" menu. Note: this is not a great option if you have changed your input/output ports or motor numbers).

Traversing a Square

In this example, we want to create a robot that will trace a square. This will be an autonomous program. This is an example of a "procedural" program in that what you are writing is an exact representation of the steps to take.

We should not try to just write the program in full right away. Instead, we should try to "build up" our program as we go.

The process we might use:

1. Think about what we want the program to do and write a simple "recipe" for it (just in words). For example:
    a. Go forward for 2 seconds.
    b. Turn right 90 degrees.
    c. Go forward for 2 seconds.
    d. Turn right 90 degrees.
    e. Go forward for 2 seconds.
    f. Turn right 90 degrees.
    g. Go forward for 2 seconds.

   h. Turn right 90 degrees.
   i. Stop.

2. Make sure that last step is there!
3. Create a program that will turn on both motors so we go forward for 2 seconds and stop. Download this program to the robot and test it out. Do not bother going on until this works.
4. Add a step that waits a little bit before starting so that we can get our hand out of the way before the robot starts moving.
5. How do we "turn right 90 degrees"?
6. We could write our complete program now; however, if we look at our "code" (often called pseudo-code) we see that steps a and b actually just repeat four times. So we can do this in a loop. Improvements like this are called "stepwise refinement"

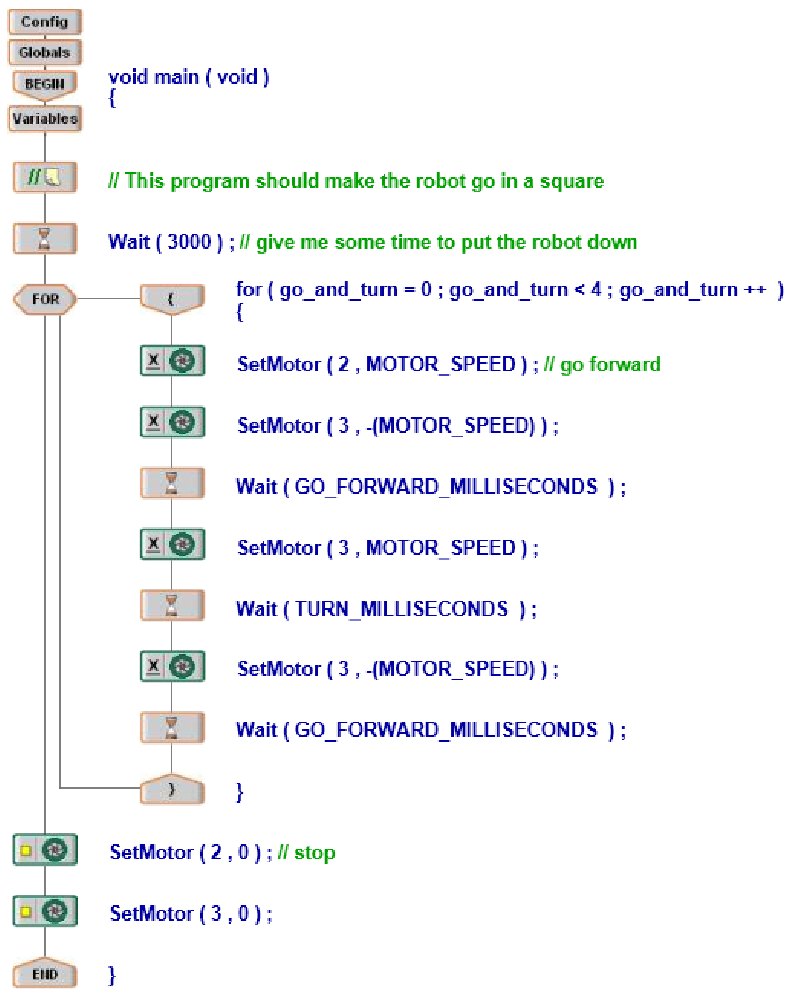
The code that is generated by easyC is shown below, and is available via the File/Print Code option.

```
#include "Main.h"
void main ( void )
{
int go_and_turn;
// This program should make the robot go in a square
Wait ( 3000 ) ; // give me some time to put the robot down
for ( go_and_turn = 0 ; go_and_turn < 4 ; go_and_turn ++ )
{
SetMotor ( 2 , MOTOR_SPEED ) ; // go forward
SetMotor ( 3 , -(MOTOR_SPEED) ) ;
Wait ( GO_FORWARD_MILLISECONDS ) ;
SetMotor ( 3 , MOTOR_SPEED ) ;
Wait ( TURN_MILLISECONDS ) ;
SetMotor ( 3 , -(MOTOR_SPEED) ) ;
Wait ( GO_FORWARD_MILLISECONDS ) ;
}
SetMotor ( 2 , 0 ) ; // stop
SetMotor ( 3 , 0 ) ;
}
```

The variables can be shown by selecting File/Print Constants & Variables:

```
Constants and Variables for project: MakeSquare
Date: 06/24/2010 Time: 10:07
-----------------------------------------------------------------------
Constants:
#define GO_FORWARD_MILLISECONDS 1000
#define TURN_MILLISECONDS 500
#define MOTOR_SPEED 127
-----------------------------------------------------------------------
Global Variables:
-----------------------------------------------------------------------
main Function:
int go_and_turn ;
```

You may also print the flowchart by using the File/Select & Print Flowchart:



```
void main ( void )
{

// This program should make the robot go in a square

Wait ( 3000 ) ; // give me some time to put the robot down

for ( go_and_turn = 0 ; go_and_turn < 4 ; go_and_turn ++  )
{

SetMotor ( 2 , MOTOR_SPEED ) ; // go forward

SetMotor ( 3 , -(MOTOR_SPEED) ) ;

Wait ( GO_FORWARD_MILLISECONDS  ) ;

SetMotor ( 3 , MOTOR_SPEED ) ;

Wait ( TURN_MILLISECONDS  ) ;

SetMotor ( 3 , -(MOTOR_SPEED) ) ;

Wait ( GO_FORWARD_MILLISECONDS  ) ;

}

SetMotor ( 2 , 0 ) ; // stop

SetMotor ( 3 , 0 ) ;

}
```

Bumper Bounce

Bumper Switches

A bumper switch is a simple digital switch with two states: pressed and not pressed. We can test the state of the switch with the GetDigitalInput () function.

Installing the Bumper Switch

Install the bumper switch in the front of the robot. For the example here, it is DI7 but you can use whatever digital input port you want.

Testing that the Bumper Switch Works

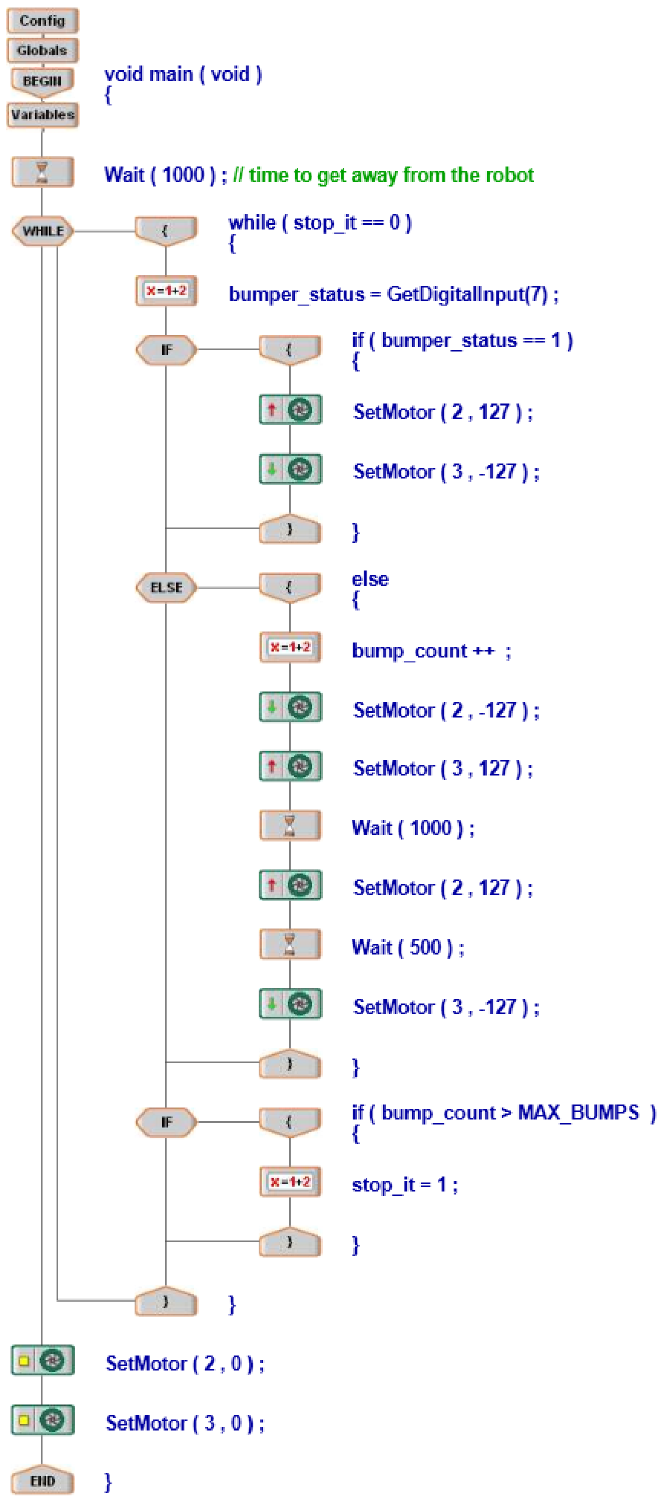
easyC provides a simple way to test the inputs and outputs of your robot. With the USB cable attached, after downloading a program, run Tools/On-line Window. The resulting dialog will allow you to monitor your robot. Click the “Enable” button to start. Then, press your bumper switch and make sure that the digital input that you expect changes state when you press it.

Sample Program

The sample program we want to write will cause the robot to go straight ahead until it bumps into something. It will then back up a bit and turn to the right a bit and then keep going in the new direction. We want it to be that we only do this 5 times.

Interacting with Sensors

When we write a program that is “event driven” (meaning that it reacts to external events), we want to normally test for that event periodically. We do this by putting all of our code in a loop so we are effectively saying “see if the bumper was hit. If so, do something. Now, start over.”.



```
void main ( void )
{

Wait ( 1000 ) ; // time to get away from the robot

while ( stop_it == 0 )
{

bumper_status = GetDigitalInput(7) ;

if ( bumper_status == 1 )
{

SetMotor ( 2 , 127 ) ;

SetMotor ( 3 , -127 ) ;

}

else
{

bump_count ++  ;

SetMotor ( 2 , -127 ) ;

SetMotor ( 3 , 127 ) ;

Wait ( 1000 ) ;

SetMotor ( 2 , 127 ) ;

Wait ( 500 ) ;

SetMotor ( 3 , -127 ) ;

}

if ( bump_count > MAX_BUMPS  )
{

stop_it = 1 ;

}

}

SetMotor ( 2 , 0 ) ;

SetMotor ( 3 , 0 ) ;

}
```

Limiting Motion

Limit Switches

You should already have two limit switches on your robot. These are digital inputs that register as ‘closed” (pressed) or “open” (not pressed).

Sample Program

We want to use the limit switches to cause the motor to stop when either limit switch is pressed, effectively preventing the motor from going too far in either direction.

This would be a very difficult and long program to write, with turning motors on and off, checking the limit switch status, etc. Fortunately, a function is built-in to easyC to do all this for us.

JoystickDigitalToMotorAndLimit

Under the Joystick function blocks is one named Joystick Digital to Motor & Limit. Using this block will do all the work we want done, but we need to fill in a lot of information.

The Program

```
#include "Main.h"
void main ( void )
{
      while ( 1 )
      {
            JoystickDigitalToMotorAndLimit ( 1 , 5 ,
                  1 , 127 , 1 , 2 , -127 , 2 , 2 ) ;
      }
}
```

Examples

easyC comes with a large number of examples, all named fairly well (in that the name of the file often describes the features used). Use them!