

Introduction

The following tutorials start with basic concepts and gradually evolve into more complex scenarios.

For additional help, intelitek recommends the following resources:

Online Help in the easyC software.

- Clicking the help buttons in the dialog boxes, menus and messages to access information specific to that software feature.
- Search for keywords
- Browse the help file index

Chief Delphi website (www.chiefdelphi.com)

Intelitek operates a forum and an FAQ on Chief Delphi to help users and receive feedback about easyC. Many experienced FIRST students and mentors use these forums to ask and answer questions, post pictures and talk about the game.

EasyC samples and test code

Many of the sensors and essential blocks in the palette also have associated sample/test code in already in easyC. To access these resources:

1. In easyC, select File > Open Project
2. Browse to the Sample and Test code folders.

Hardware Configuration

The tutorials provided below are based on the kitbot chassis included in the kit of parts. You can follow along with any system of your choosing, however you will need to adapt the motor ports and sensors in the examples to fit your particular setup. Please connect the sensors and motors as follows:

Operator Interface Setup

- Connect a standard joystick to Port 1
- Connect a Dongle to the competition port on the Operator Interface.
- Create a radio or tether link with your robot

Motor Setup

- Connect the left motor/ transmission to PWM Port 1
- Connect the right motor/ transmission to PWM Port 2

Gear Tooth Sensor

- Wire the gear tooth sensor following the instructions provided by IFI.
- Mount a gear tooth sensor inside each of the 2006 kitbot transmissions, or mount an encoder to the end of the driveshaft if using the 2007 banebot transmission.
- Connect the left gear tooth sensor to Digital Input Port 1
- Connect the right gear tooth sensor to Digital Input Port 2

Gyro

- Wire the gyro following the instructions provided by IFI.
- Mount the gyro in the middle of the centerline of the rear drive wheels. This provides the most accurate feedback.
- Connect the gyro to Analog Input Port 1

Connect the Camera

Read and understand the camera documentation available for the CMU Camera before attempting to use it on your robot. Please refer to your camera documentation for information about wiring your CMU Camera. See also the easyC Pro online help.

The easyC Interface

easyC Pro is a single platform that supports programming with both the Vex and FRC controllers. You can switch between these controllers using the Robot Controller Setup in the Options Menu. Keep in mind that although the two sides of easyC appear very similar, the code generated by each is unique. Just as you cannot open Vex code from the FRC side of easyC, neither can you load Vex code onto an FRC robot.

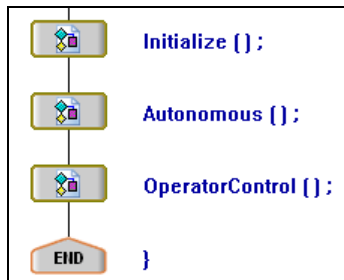
You will spend most of your time programming in the Block Programming Window (the flowchart). To the right of this window is the C Code Window, where all C Code generated by your blocks is shown. The Project Explorer has two tabs, the Function Block Tree, and the Project Tab. The Project Tab is where many of the new and advanced features of easyC Pro live, including text editing, and Source File inclusion. You will convert a block function to C Code near the end of these tutorials, and familiarize yourself a little with these features.

Competition Project

A Competition Project is the framework in which you will write your code for a FIRST competition. When you put your robot on the field at a competition, it is important that your code be fully compatible with the field control. Competition Projects make it simple to write a program which is fully compatible by hiding all of the necessary controls in the background, and providing you with clearly marked locations for your autonomous and operator controlled code.

A competition project is also smart and flexible, allowing you to initialize devices before the autonomous round begins, and automatically sensing the beginning of the autonomous and operator periods.

When you open a new competition project, the block programming window will contain three functions; **Initialize**, **Autonomous**, **Operator Control**.



The **Initialize** function is the best place to initialize your sensors. The code you write inside the Initialize function will execute as soon as the robot is turned on and will continue to execute until completed. When the Initialize code has finished running, the robot will automatically wait for the autonomous round to begin, and will call the autonomous function when the autonomous round begins. This will happen behind the scenes; you don't need to write any special code for this to happen.

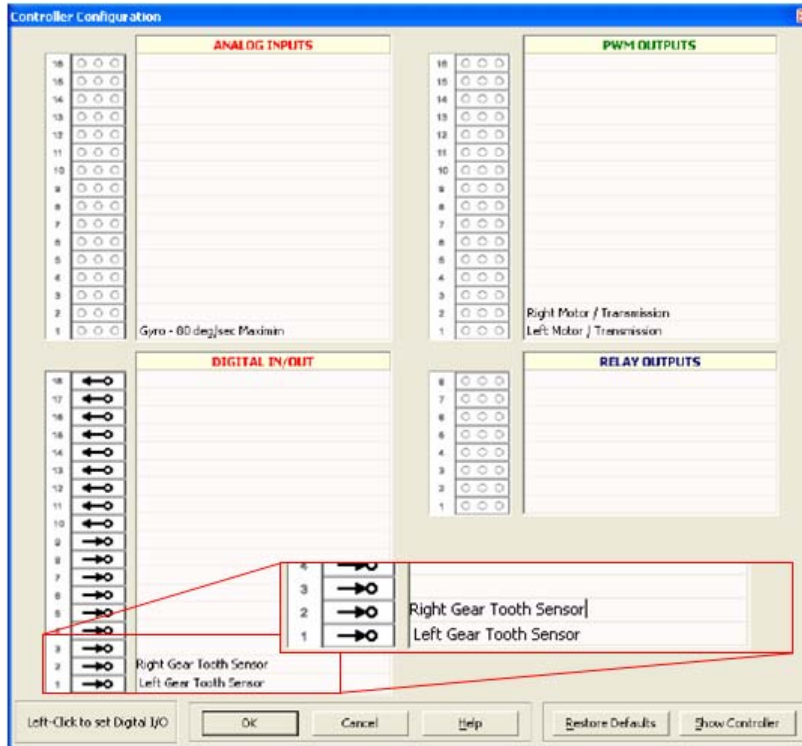
When the field control switches to enabled and autonomous (the chart below), your program will begin to execute the code you place in the **Autonomous** function. Your autonomous code can contain anything from wait statements to loops. Unlike the Initialize Function, when the field control switches states from autonomous to operator control, your autonomous code will automatically terminate. You do not need to write any special code for this to happen. In Autonomous, you may not send any signals from the OI to the Robot Controller.

When the autonomous round ends, the program will automatically call the **Operator Control** function. In Operator Control you are able to send signals to the robot controller from the OI. Place a repeating loop of some sort in the Operator Control function to continuously sample the data being sent from the joysticks over the OI.

Using the Controller Configuration

Once your hardware is properly configured, you should enter the configuration (your robot map) into easyC. A configuration stored in easyC can always be referenced if you forget where a device is connected by printing the robot map, or hitting the F5 key to open the dialog. The descriptions you enter are visible in the On-Line Window, and in the description fields of many of the dialog boxes in easyC.

1. Open the Controller Configuration dialog box by pressing the F5 key.
2. Enter the details about your robot. If your kitbot is configured according to the guidelines on page 1, it should look something like the example below. 1.



Testing with the On-Line Window

The On-Line Window is a powerful testing tool used to troubleshoot and understand the nature of your robot's hardware. The On-Line Window can also be used to view feedback from the sensors connected to the controller. The window can display if a limit is tripped or the analog feedback from inputs like a potentiometer.

Configure your system for downloading:

1. Select Build & Download > Loader Setup.
2. Verify the number of the COM port that the serial cable is plugged into. The other end of the programming cable should be connected to the programming port on the FRC controller.
3. Select the Terminal Window radio button to launch the terminal window after downloading the program. This will be used in the later tutorials.
4. Select **OK** to save the settings.
5. Press and hold the **PROG** button on the FRC controller until the Program State LED turns orange.
6. Prop up the wheels of the robot so that they are not touching the ground. This is a safety precaution and a good practice to get in to when working on the robot.

Download the On-Line Code

1. Select **On-Line Window** from the Build & Download Menu or select the On-Line Window icon in the main toolbar.

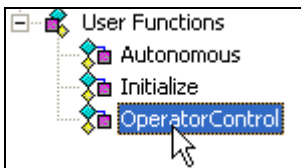


2. Click on the **Download On-Line Code** button. This will bring up the built in loader. As the bar moves left across the screen, it is erasing the memory of the controller and as the bar moves to the right, it is loading the new project.
3. Verify the Competition Dongle plugged into the Operator Interface (OI) is set to Enable and Operator Control (not Autonomous).
4. Move the PWM Output #1 slider slowly to the left and to the right. Watch the left motor of the robot rotate back and forth.
5. Do the same for the right motor. The description fields in the Online-Line Window are fully editable. Enter what you just learned, i.e. write down the values that make each motor go forward into the description for that motor.
6. If you have an accelerometer or other sensors connected, add them to the list. To add a description for a new sensor, click **Show All**, enter your description in the appropriate location, and click **Show Defined** to shrink the list back to normal.
7. Click **Save**.
Your Robot Configuration in easyC has now been updated with the new changes. You should see the numbers changing for the analog/digital ports to which your sensors are connected. For digital and simple analog devices, these numbers are the actual returned values. For more complicated sensors like the Gyro, that need to be initialized, the online window will tell you if the sensor is active, although the values themselves are not valid.
8. Close the window when your testing is complete.
9. Save your project. You can import your robot map from this project in the future.

Programming Operator Control:

A FIRST robot must move around the field under the control of a radio operator. The amount of time a robot will spend in Operator Control is historically much greater than the time the robot will spend autonomously. Programming the robot to respond to commands from the Operator Interface (transmitter) is a simple task requiring only a few lines of code.

1. Select File > New Competition Project.
2. Select Options > Import Controller Configuration and select your saved project from earlier.
3. In the Project Explorer, double-click on the Operator Control function in the User Functions group.



4. Click and drag a While Loop function block from the Program Flow group in between the Variables and End blocks.
5. Type the number '1' in the expression box and select OK. This will make the While Loop run forever allowing the information in the controller to be constantly updated from the input on the Operator Interface (OI).
6. Click and drag an Arcade – 2 motor function block from the RC Control group in between the brackets of the While Loop.
7. Enter the number '1' to specify the Port for both channels.
8. Enter the number '2' (Y Axis) for the Forward/Reverse OI Channel, and the number '1' (X axis) for the Rotate Channel.
9. Enter the number '1' for the Left Motor and the number '2' for the Right Motor.

Arcade - 2 motor

Forward\Reverse Operator Interface Channel:
 Port #: 1 (Value Range: 1..4)
 Axis #: 2 (1 = X-Axis, 2 = Y-Axis, 3 = Wheel, 4 = Aux Analog)

Rotate Operator Interface Channel:
 Port #: 1 (Value Range: 1..4)
 Axis #: 1 (1 = X-Axis, 2 = Y-Axis, 3 = Wheel, 4 = Aux Analog)

PWM (Value Range: 1..16):

Left Motor:
 1
 // LeftMotor 0=forward
 Invert Direction ☐

Right Motor:
 2
 // RightMotor 0=backwards
 Invert Direction ☐

Code:
 Arcade2 (1 , 2 , 1 , 1 , 1 , 2 , 0 , 0);

Comment:

F6 – Globals and Constants Ctrl + F6 – Local Variables

OK Cancel Help

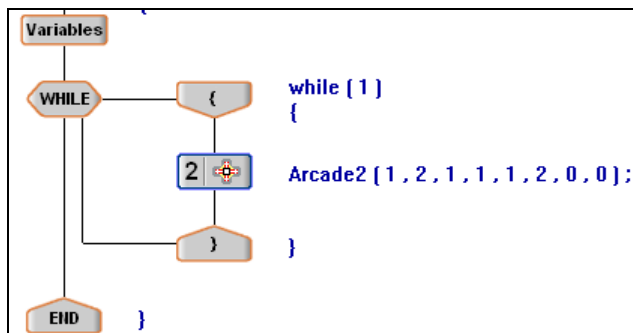
Notice the descriptions that appear in the Arcade Block below each PWM selection.

10. Select OK to accept all the defined parameters.
11. Verify the Competition Dongle plugged into the Operator Interface (OI) is set to Enable and Operator Control (Not Autonomous).

Note:

Before downloading the program to the robot, remember to elevate the robot so the wheels are not in contact with the floor.

12. Select Build and Download from either the Build and Download menu or the icon on the main toolbar.
13. When the download is complete, move the joystick back and forth and verify the drive motors move accordingly.



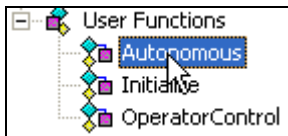
Autonomous Programming

The next several tasks will guide you through autonomous programming. With practice and creative thinking, autonomous programming will no longer appear as daunting.

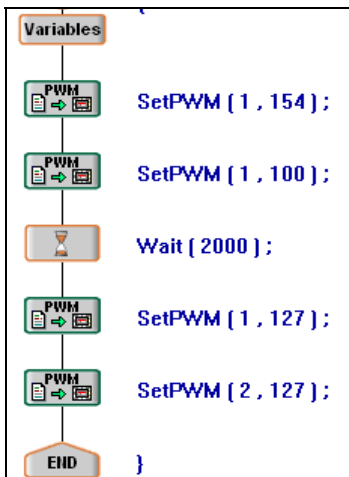
The first task you will accomplish is:

Drive straight and stop

1. Select File > New Competition Project.
2. To work with the robot map you wrote earlier, select Options > Import Controller Configuration and select your saved project.
3. Double click on the Autonomous function in the User Functions group in the Project Explorer.



4. Click and drag a PWM Control function block from the Output group in between the Variables and End blocks.
5. Set PWM #1 (Left Motor) to '154' and select OK.
6. Drag in another PWM block and set PWM #2 (Right Motor) to '100'. Depending on the polarity of your motors, these values should drive your robot forward. You may have to reverse which values are sent to each motor, however.
7. Select and drag a Wait function block from the Program Flow group after the two PWM Control blocks.
8. Enter '2000' for the wait value and select OK. The value for the wait function is in milliseconds. Your robot will continue driving forward for 2 seconds.
9. Select the first PWM Control block, hold down the 'shift' key and select the second PWM block. Right click and select Copy. Right click on the line below the Wait function block and select Paste. This will copy both blocks in one step.
10. Double click on each of the new PWM Control blocks. Change the Set value to '127' for both motors. A PWM value of 127 stops the motor.



11. Press the PROG button the FRC controller until the Program State LED turns orange and then release the button. Verify the wheels of the robot are not touching the ground.
12. Select the Build and Download from either the Build and Download menu or the icon on the main toolbar.
13. Verify the Competition Dongle is set to Enable and Operator Control (Not Autonomous). After the program completes its download, switch the Dongle to Autonomous to execute your program. The left and right motors should turn on for 2 seconds and stop.

Autonomous Driving with the Drive Library

An easyC library is a complete reference information archive. It consists of User Functions, Global Variables and Macros; in short, everything you need to integrate your imported code into any project.

In this tutorial, you will import a library of drive functions provided for you in easyC, and learn how they operate. These drive functions simplify the process of activating PWM's, by shifting the neutral value to zero, and associating forward speed with positive values up to 127, and reverse speed with negative values up to -127. You can also combine turning with driving in the same function, by specifying a direction along with the speed.

You will find that libraries are helpful tools, and that sharing libraries with other friends or teams is a good way to increase productivity, and help others in need.

1. Select File > New Competition Project.
2. To work with the robot map you wrote earlier, select Options > Import Controller Configuration and select your saved project.
3. Select Import Library from the Options menu. Read the description of the library to learn a little more about it.
4. Select the Library Drive Functions.ECL and click on the Add button. All of the user functions and variables in the library are now added to the User Functions group, and to your Global Variables list. Close the dialog box.
5. Double click on the Autonomous function block in the User Functions group in the Project Explorer.
6. Click and drag a TwoWheelDrive function block from the User Functions group in between the Variables and End blocks.
7. Click into the Argument block of the left motor and enter the number '1'. This defines the left motor to be plugged into PWM port #1. Enter the number '2' for the right motor PWM.

Function: TwoWheelDrive

Retrieve to:
void

Define Arguments:

#	Type	Name	Argument
1	unsigned char	_leftmotor	1
2	unsigned char	_rightmotor	2

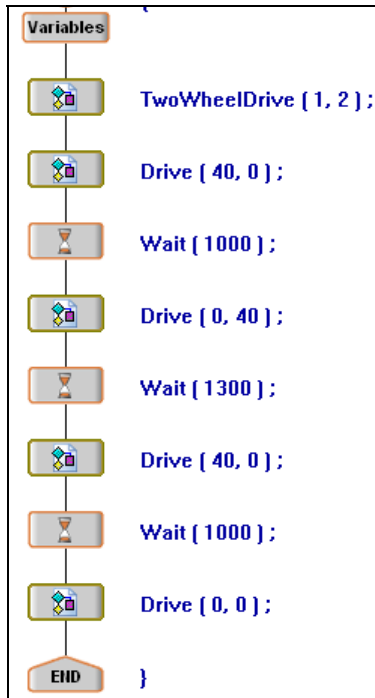
Code:
TwoWheelDrive (1,);

Comment:

F6 - Globals and Constants Ctrl + F6 - Local Variables

OK Cancel Help

8. Click and drag a Drive function block under the TwoWheelDrive function.
9. Enter '40' for Speed and '0' for direction. This will drive the robot slowly forward.
10. Drag in a Wait function block. Leave the wait time at 1000 ms.
11. Drag in another Drive function block to turn the robot. Enter '0' for Speed and '40' for direction. This will spin the robot counterclockwise in place.
12. Drag in a Wait Function block for 1300 ms. The robot will continue spinning for this long.
13. Copy the first Drive function block by selecting it while holding down the 'ctrl' key. Drag a copy of the Drive function after the last Wait block. The robot will drive straight again.
14. Insert another Wait function for 1000 ms.
15. Insert another Drive function and set both the Speed and Direction to '0'. This will stop the robot.



The program you have just written will drive the robot forward, turn it, drive it forward again, and finally stop.

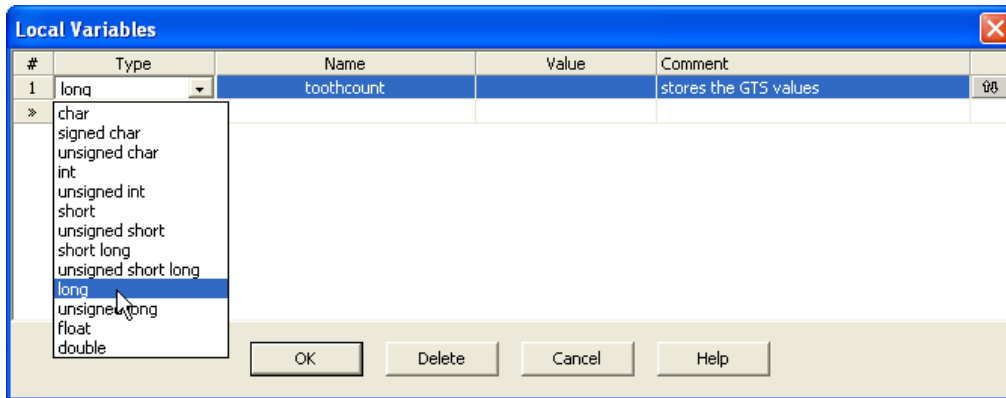
- Download the program to the controller and test it. The robot's behavior may vary depending on the design of the drive system, polarity of the motor, level of the battery and friction in the system.

Adding Sensor Feedback to your Autonomous Code

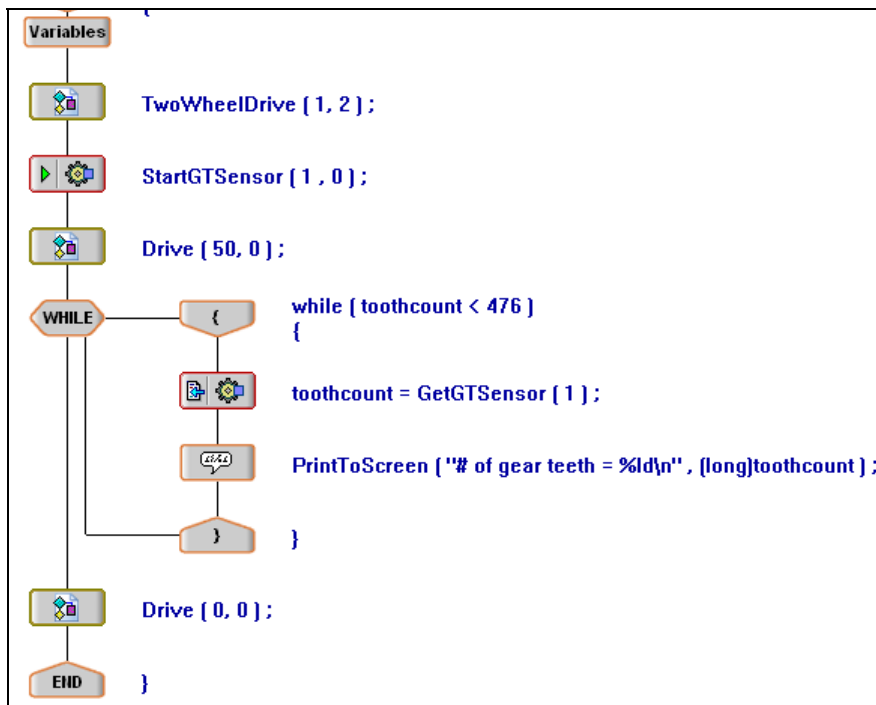
Learning to gather and properly interpret sensor data is critical. Using the wrong sensor for a task, sampling a sensor at the wrong time, or sampling too quickly, can lead to poor performance at the competition. Learn how your sensors work, and always run simple tests to confirm their functionality before proceeding too far into a design.

This tutorial will show you a simple way to control how far you have moved by sampling the gear tooth sensor provided in your kit of parts. The sensor is best (and most easily) mounted in the 2006 kitbot transmission. Alternatively, any encoder connected to a drive shaft can be substituted for the gear tooth sensor (but remember to use the encoder blocks instead).

- Select File > New Competition Project.
- To work with the robot map you wrote earlier, select Options > Import Controller Configuration and select your saved project.
- Import the Drive Libraries and configure your two wheel drive system as you learned to do in the previous tutorial.
- Drag in the user function Drive, and set it to '50,0' to set the robot in motion.
- Drag a Gear Tooth Sensor Block from the Inputs group before the Drive block. Select the Start option and define the interrupt port # as '1'. Leave Invert Direction normal (0). Select OK to finalize the block.
- Double click on the variable block at the top of the flow chart. Local variables are defined in the dialog.
- Click in the type box and select Long from the drop down menu. Name the variable "toothcount" and select OK.



8. Drag in a While loop under the PWM Control block.
9. Use the Add Variable arrow to view a list of already defined variables and constants. Select the toothcount variable and note how it is automatically added to the expression of the While Loop.
10. To complete the expression type in a less than sign (<) and the number '476'.
11. 476 is twice the number of gear teeth the sensor will see in one rotation of the wheel using the standard IFI gearbox, chain and sprocket.
12. Drag a Gear Tooth Sensor block into the While Loop. Select the **Get** option and define the interrupt port as # '1'. Select the toothcount variable from the drop down list to store the information from the sensor. Select **OK** to finalize the block.
13. Drag a Print to Screen block from the Program Flow group under the Gear Tooth Sensor block in the While Loop. Enter '# of gear teeth = ' in the message box.
14. Select **toothcount** from the Variable list. This will print the value of the toothcount variable to the terminal window as the program executes.
15. In the Directive field, select **%ld** in the list. This displays long variables. In the Type-cast list, choose **long**. Click **OK** to finish.
16. After the While Loop, add a Drive Block to turn off the motors (set both speed and direction to zero).



17. Download the program to the controller.
- The kitbot should drive forward for two revolutions of the wheels. The value of the gear tooth sensor will be displayed in the terminal window. The wheels should lose power when the sensor has counted 476

teeth, however Inertia and momentum may cause your results to vary. Applying stiff resistance to the wheel as it turns will give you better feedback.

Using the Graphic Display to test the Camera

In this tutorial you will learn how to use the Graphical Display Block in easyC. A representation of the camera view, including detailed information about the target light will be displayed in the Graphic Display Window. easyC makes it simple to get the camera quickly online and responding to your commands.

1. Ensure that your camera is connected correctly to your robot controller.
 - easyC assumes your camera is connected to the TTL serial port located on the robot controller. This connection is generally made using the TTL-Serial adapter board provided by Innovation First.
 - To use the 3 pin RS232 port on your camera, be sure nothing is connected to the 9 pin RS232 Port, or to the TTL Port.
 - Your camera power should be connected to an unused PWM Port and is powered like a servo by the backup battery. It is especially important to monitor the power level of your backup battery, as the camera will function erratically if not properly powered. See your camera documentation for more information. Consider IFI alternate power supply circuit.
2. Select File > Open Project and browse for the GDCamera_FRC.ecp file in the Test Code folder. This project is already written and is designed to display a representation of the camera's view in the Graphical Window.
3. Download the test code to the controller.
4. When you are finished downloading, open the Terminal Window if it hasn't opened automatically.
5. Click the Graphic Display checkbox to activate the Graphic Display feature.
6. Point the camera at the light. A rectangle representing the outer boundary of the target, and an 'X' marking the centroid are displayed on the screen. Additional values are displayed in text below.
7. Move the camera around and notice how the target and centroid change position in the window.

Note:

It is possible for the test program to act sluggish and non responsive on your system. Sending too much information to the Graphical Display Window can cause the buffer to overflow on some systems. The allowable data upload rate will vary depending on your system speed and serial port or USB port speed. Ideally, you should use timers to control the upload rate if it becomes a problem. Using a timer, you can display a string of graphical blocks every quarter or half second, without disrupting the speed at which the remainder of your program operates.

Driving with the Camera

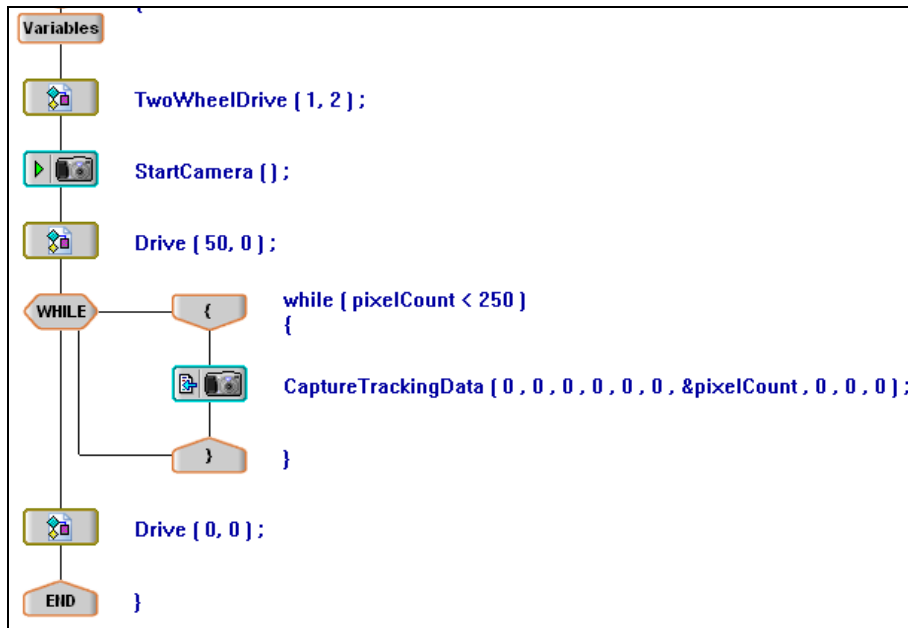
The camera is essential to the FIRST competition. easyC makes it simpler than ever before to get the camera to quickly respond to your commands. This tutorial will guide you through a simple program that will make the robot drive straight until it nears the green target light.

Note:

Be careful when running this program and always have a spotter prepared to disable the robot in an emergency.

1. Select File > New Competition Project.
2. To work with the robot map you wrote earlier, select Options > Import Controller Configuration and select your saved project.
3. Import the Drive Function.ECL library.
4. Double Click on the Initialize function of the Competition project to open it
5. Drag an Initialize function block from the Camera group into the Initialize function of the project.
6. Click on the Setup button. This is where you can import or export sets of camera information. All the tables have been defaulted to values that will work for the illuminated green target. Select OK to accept all the defaults and select OK to insert the block.
7. Drag in a TwoWheelDrive block into the Initialize function of the project and define the left motor as PWM '1' and the right motor as PWM '2'.
8. Double click on the Autonomous function block.
9. Drag in a Start function block from the Camera group and select OK.
10. Define a variable of the type unsigned char named 'pixelCount' in the Variable block. All the variables stored in the camera structure must be of the type unsigned char.

11. Turn the motors on so the robot will drive slowly forwards with a Drive command.
12. Drag in a While Loop that will execute as long as 'pixelCount < 250'.
13. Drag a Capture function block from the Camera group into the While Loop.
14. Select the variable 'pixelCount' from the drop down list to define the Pixel Density value. This value is an indication of the strength and clarity of the target image. This number increases as the camera and target near one another.
15. Add a Drive Block after the While loop to stop the motors.



16. Download the program to the controller and test your program.

The robot should drive straight until the Green illuminated target is very close. Adjust the Pixel Count value to fine tune the distance. Tutorial 8 in the easyCPro, FRC Help File includes a more advanced camera tracking tutorial.

Drive straight with the Gyro

In a previous tutorial you wrote a program that would drive a distance and stop. In this tutorial, you will use the Gyro to write a program that incorporates active feedback.

The drive systems on most robots are not equal. This is especially true on the kitbot because the motors mounted to one of the gearboxes are mounted in reverse, and will have different power characteristics.

The program you will write is simple but powerful enough to correct all but the most severe drive system imbalances. Your robot should drive flawlessly to the desired heading, in this case a straight line, every time.

First it may be helpful to test your Gyro and see what values it returns.

Testing the Gyro

1. Select File > New Competition Project.
2. To work with the robot map you wrote earlier, select Options > Import Controller Configuration and select your saved project.
3. **Initialize** and **Start** the Gyro in Analog Port # '1' in the Initialize function.
4. In the Autonomous Function, define a local integer named 'heading'.
5. In a While Loop, **Get** the Gyro Angle and return it to the variable you just created.
6. Print the heading value to the Graphical Display using a Graphic Display block. Select any convenient coordinate such as row '5', column '5'. The graphic display will continually update the value rather than stream the number like the terminal window would do.
7. Add a wait statement of 200 ms, and clear the text region where you printed your previous values with another Oblock.
8. Download this code to your robot controller. Remember to switch to the Graphic Display in the terminal window to see the output.

- Observe and record the values of the Gyro at different angles.

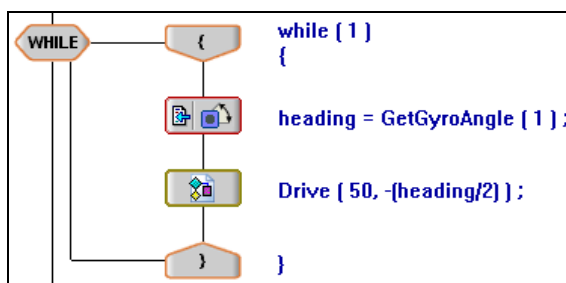


Writing an auto-correcting program

- Delete the Graphic Display blocks and Wait statement from the example above.
- Select **Import Library** from the Options menu.
- Select **Drive Functions.ECL** and click on the **Add** button to close the dialog box.
- Double Click the Initialize function of the Competition project in the explorer to open it.
- Drag in a TwoWheelDrive function block and define the left motor as PWM # '1' and the right motor as PWM # '2'.
- Double click on the Autonomous function block to open it.
- Drag a Drive function block under the Print to Screen block. Enter '50' for the speed.
- Click into the Direction field and select **heading** from the drop down menu placing a negative sign (-) in front of the variable. This will allow the robot to correct itself in the proper direction (depending on your motor polarities, you may not need a negative sign).
- Download the program to the controller to test it.

The robot will use the feedback from the gyro to power the left and right motor proportionately to keep it driving in a straight line. If your robot overshoots the line with every correction, the heading value from the gyro may be too big to use directly as the compensation value. To reduce the oscillation, we need to introduce a correction factor.

- Open the Drive function block in the While Loop.
- Divide the negative Heading value by 2. This will reduce the amount of correction and allow the robot to drive in a straighter line. The value for direction should now be `"-Heading/2"`.



- Download the program to the controller and test again.

The robot should now drive straighter and correct its course faster. Try pushing the robot off course and see if it can correct.

Any field in easyC can be made into an expression. Simple math operations like adding, subtracting, multiply and dividing can be very useful when trying to process feedback from sensors.

Advanced Features

Converting block functions to C Code allows you the freedom to edit the code yourself. In this example, you will write a function using the **Interrupt Watcher**, convert that function to C Code, edit it, and finally download the program to the robot.

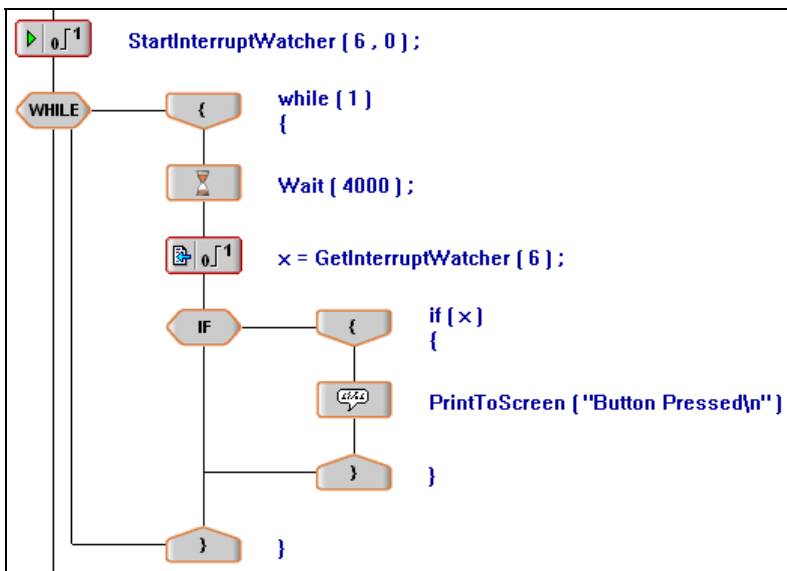
The Interrupt Watcher is a function block that is new to easyC Pro, but existed previously as part of the WPI Library. Many other convenient features and functions exist in WPI Library, and are accessible in easyC. Some of these functions include a battery life Indicator and a system clock. The prototypes for these functions are found at the end of the API.h header file.

For this example, you will need to connect some type of digital switch, such as a limit switch, to digital input port 6 on the Robot Controller.

1. Select File > New Standalone Project.
2. Right click on User Functions in the Block Tree and select Add New Function.



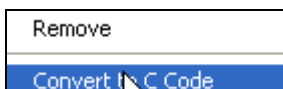
3. Give the New Function a name, for example 'My_Watcher'.
4. In your new User Function define an unsigned character variable named 'x'.
5. Drag in an Interrupt Watcher block from the inputs group, and set the interrupt port to Port #6. Leave the direction field zero (0).
6. Create an infinite While Loop.
7. Within the While Loop wait 4 seconds, get the value of the Interrupt Watcher and return it to the variable 'x', and add an If statement that checks 'x' to see if it is true.
8. If 'x' is true, print "Button Pressed" to the screen.
9. Drag your function from the palette into Main (which should be empty). All that Main will do is call your function a single time. Because your function contains an infinite while loop, your program will still run forever.



10. To convert this function to C Code, switch to the Project Tab in the Explorer.



11. Right click on the function name, and select 'Convert to C Code'.



A message will appear asking if you are sure you want to continue, and reminding you that any existing function calls (such as the one you played in Main) will be converted to equivalent C Code.

12. Click **Yes**.
 13. Your function is now located under Source Files in the Project Tab, and has been removed from the Block Tree. To call the function in the future using easyC blocks, you will need to use the User Code block. The contents of this block would be identical to the User Code Block that now appears in Main.
- We missed a line of code the first time through and can add it now.
14. Create a blank space using the enter key on the line above the Wait statement.
 15. Return to the Function Blocks tab, and drag a PrintToScreen block into your C Code on the new line you just created. You can drag blocks into the C editor just as you can the Block Window. Have the text 'Waiting' print to screen.

```
U User_Watcher.c
#include "Main.h"

void User_Watcher ( void )
{
    char x;

    StartInterruptWatcher ( 6 , 0 ) ;
    while ( 1 )
    {
        PrintToScreen ( "Waiting\n" ) ;
        Wait ( 4000 ) ;

        if ( 1 )
        {
            PrintToScreen ( "Button Pressed\n" ) ;
        }
    }
}
```

Before you compile the program, there is one thing left to do. When you converted your User Function to C Code, the prototype or definition for the function was deleted from Main.h and must once again be listed in a header file.

16. Open UserInclude.h from the list of header files in the Project Tab.
17. Delete the comments about adding code, and in that line, write the function definition so it looks like the example below. Don't forget the semi colon at the end of the line.

```
h UserInclude.h
// UserInclude.h : header
#ifndef USERINCLUDE_H_
#define USERINCLUDE_H_

void User_Watcher(void);

#endif // USERINCLUDE_H_
```

18. Compile and Download the code.
- If you quickly press and release the digital switch you have connected to port 6 during the four second wait period, the Interrupt Watcher will catch this state change, and print the message 'Button Pressed!' to the screen. If you do not hit the switch, nothing will happen. You can use the Interrupt Watcher to catch very fast state changes on the interrupt port that may happen very quickly, or at times when your program is busy doing other things (like waiting in this case).