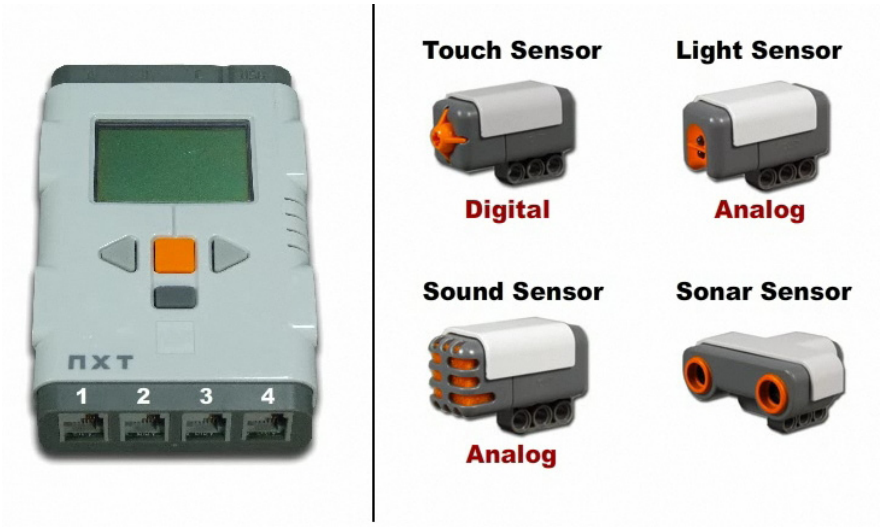
TETRIX™ Sensors

TETRIX Controllers

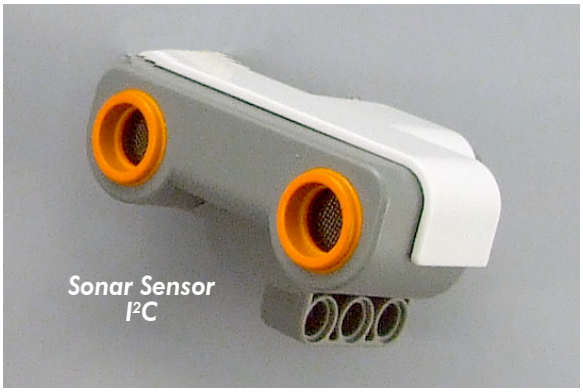
In this lesson you will learn how the HiTechnic Motor Controller and Servo Controller work with your NXT by using a Sensor Port.

How do sensors communicate with the NXT?



On the NXT there are four sensor ports, which are typically used for the four NXT Sensors: Touch Sensor, Light Sensor, Sound Sensor, and Sonar Sensor.

The NXT Sensors are usually classified as either analog or digital. For both classifications, the NXT brick does all the processing necessary to interpret sensor data. The Light Sensor and the Sound Sensor are both analog. The Touch Sensor is digital.

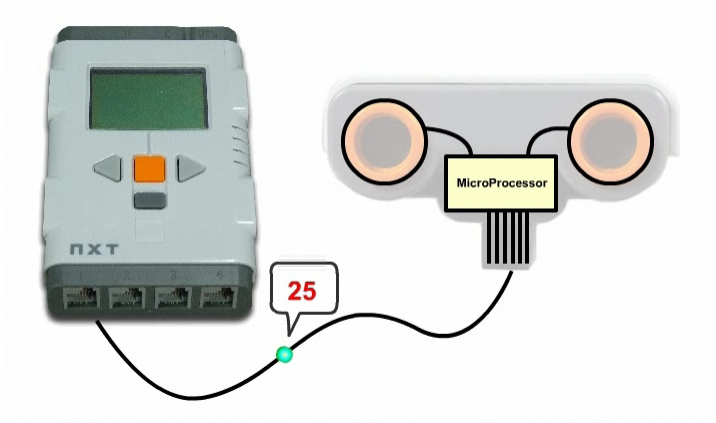


The NXT Sonar Sensor is in a different classification of sensor. The Sonar Sensor is an I^2C Sensor.

I^2C stands for Inter-Integrated Circuit. I^2C is a type of serial messaging that allows more information to be sent though a cable.

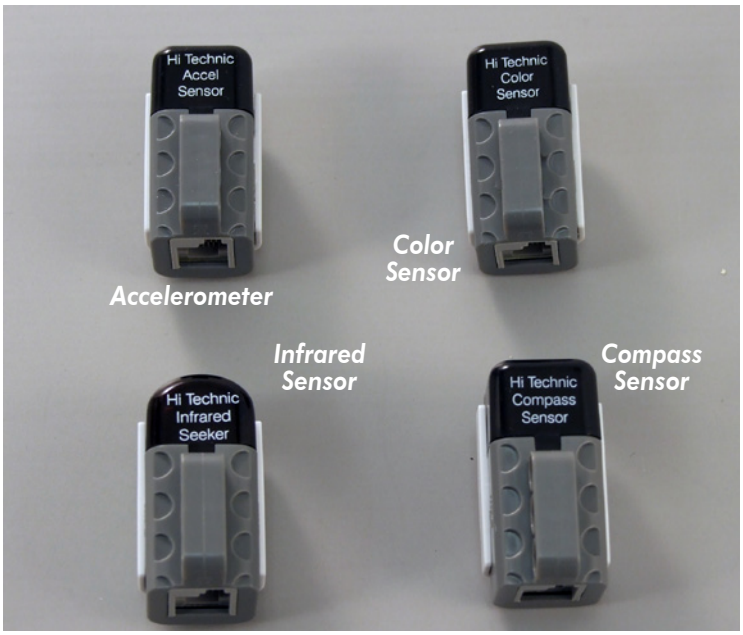
TETRIX™ Sensors

TETRIX Controllers (cont.)



Unlike simple analog and digital sensors, I²C sensors contain a microprocessor inside of the sensor that processes the data before it gets to the NXT.

For example, inside the Sonar Sensor there is a microprocessor that reads the data and processes it, and returns a value in centimeters to the NXT over the I²C bus.



I²C compatibility is useful because it expands the capabilities of the NXT to accept a wide range of third party sensors and devices.

Some examples of third party sensors include the compass sensor, the accelerometer, the color sensor, and the infrared seeker.

TETRIX Controllers (Cont.)

Two examples of I²C devices are the HiTechnic Motor Controller and the Servo Controller.

Due to I²C protocol the NXT can control additional motors without using a motor port.

DC Motor Controller

Servo Controller

I²C makes it possible for the NXT to control the speed of 2 motors and the positions of 6 servos.

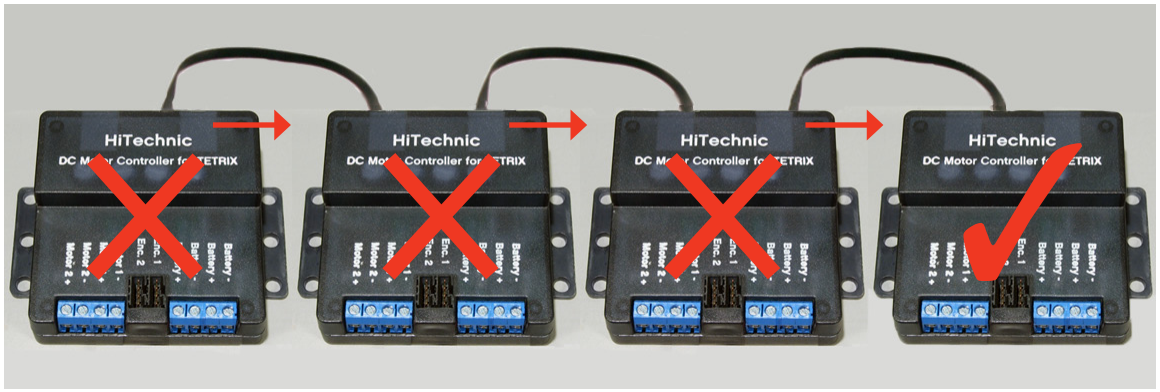
Also, I²C is used to read encoder values from the motor controller.

Another useful feature of the controller is the ability to daisy chain, which allows you to connect up to four controllers to a single sensor port. Daisy chaining is connecting multiple controllers together.

TETRIX Controllers (Cont.)

Daisy chaining works by having the first controller in the chain read all the messages sent out by the NXT. The controller will pass along all the messages not intended for that specific controller.

The messages are then sequentially passed along until they reach their final intended destination.



If the NXT sends the message that the "final" DC Motor Controller should turn on its motors, the message must reach that motor controller

---

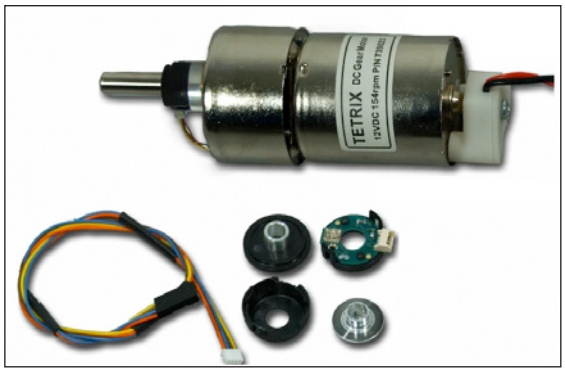
Each controller is assigned a unique address to ensure that the messages sent out get to the correct controller.

However, this system is only as effective as the programming behind it. If the controllers are not properly configured the messages that are sent out to the controllers will end up at the wrong destination and the robot will not behave as intended.

S1_C1 *S1_C2* *S1_C3* *S1_C4*

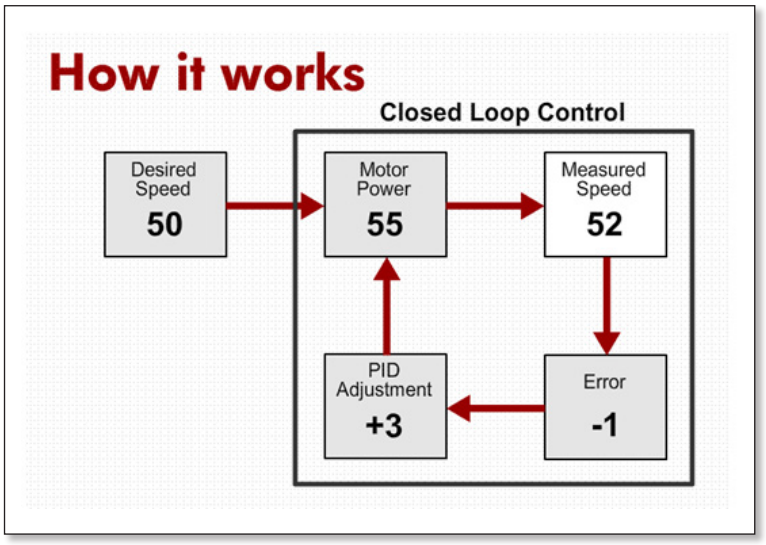
TETRIX Encoders

In this lesson you will learn how to use the TETRIX DC Motor Encoders.



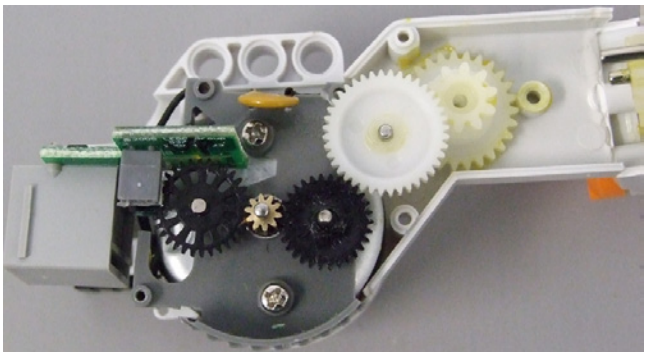
The best way to improve control of TETRIX DC motors is to install TETRIX encoders.

The rest of this lesson assumes that you have already installed the TETRIX encoders. If you have not yet installed the encoders, do so now.



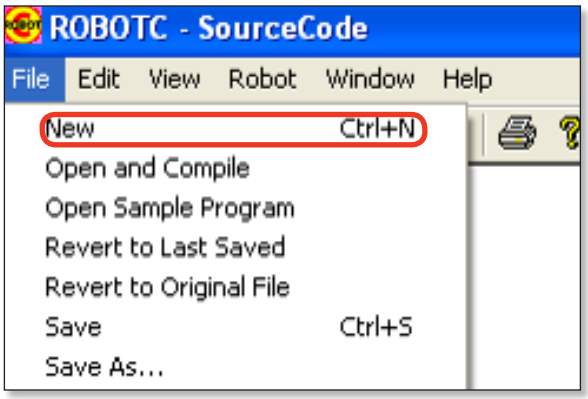
In the NXT-only Improved Movement section, PID was one of the topics discussed. The PID algorithm monitors the rate of movement of a motor and adjusts the speed of the motor automatically.

Therefore, when a motor meets more resistance the PID algorithm increases the “actual” power level to meet the “requested” power level.



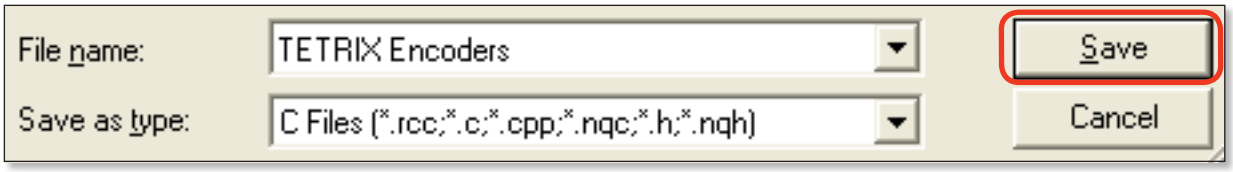
The NXT motors have built-in encoders, therefore PID is enabled by default. The TETRIX DC motors do not have PID enabled by default because they do not have built-in encoders. With the TETRIX DC motors PID needs to be manually enabled.

TETRIX Encoders (cont.)

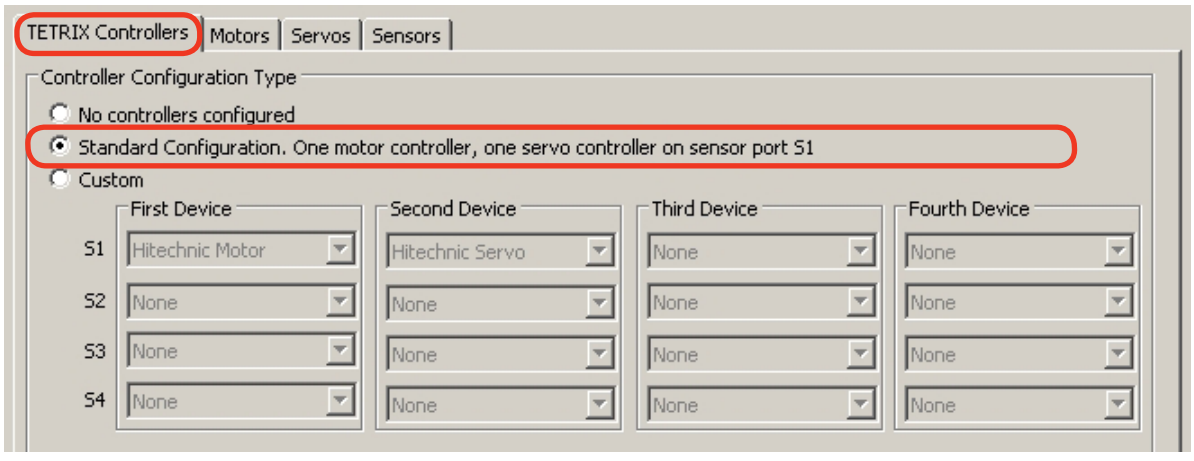


1. Go to the "File" menu.
2. Click "New"

3. Go to the "File" menu.
4. Click "Save".
5. Save your program under the name "TETRIX Encoders".



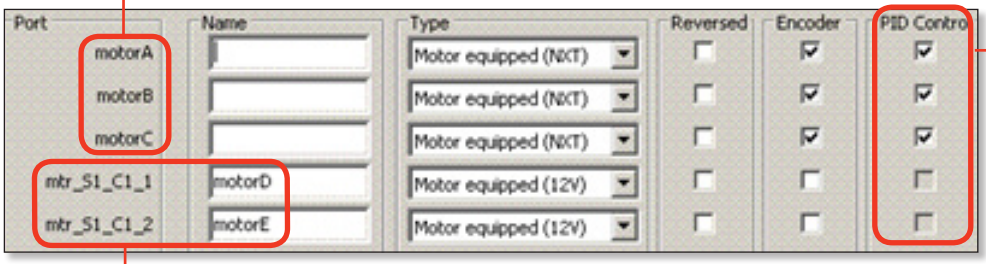
6. Go to the "Robot" menu.
7. Click "Motors and Sensors Setup"
8. Go into the "TETRIX Controllers" tab
9. Choose "Standard Configuration".



TETRIX Encoders (cont.)

Select "Motors" tab.

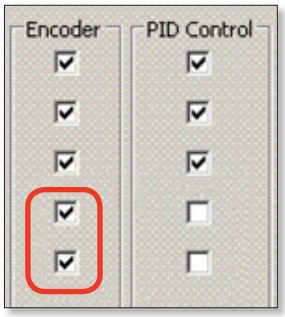
The three motors at the top of the window are the NXT motors; named motorA, motorB and motorC.



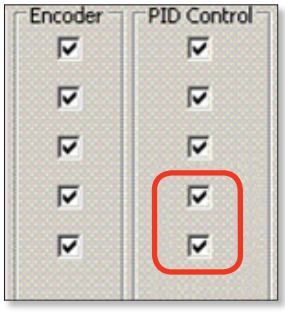
In the PID column, the first three motors are already checked to have PID enabled, because PID control on the NXT motors is automatically enabled.

The TETRIX motors do not have PID Control enabled at this point.

The two motors at the bottom are TETRIX motors; named motorD and motorE.



Click on the encoder checkboxes next to motorD and motorE to inform the robot that the encoders should be enabled. Note once you do this the PID checkboxes for these two motors are no longer grayed out.

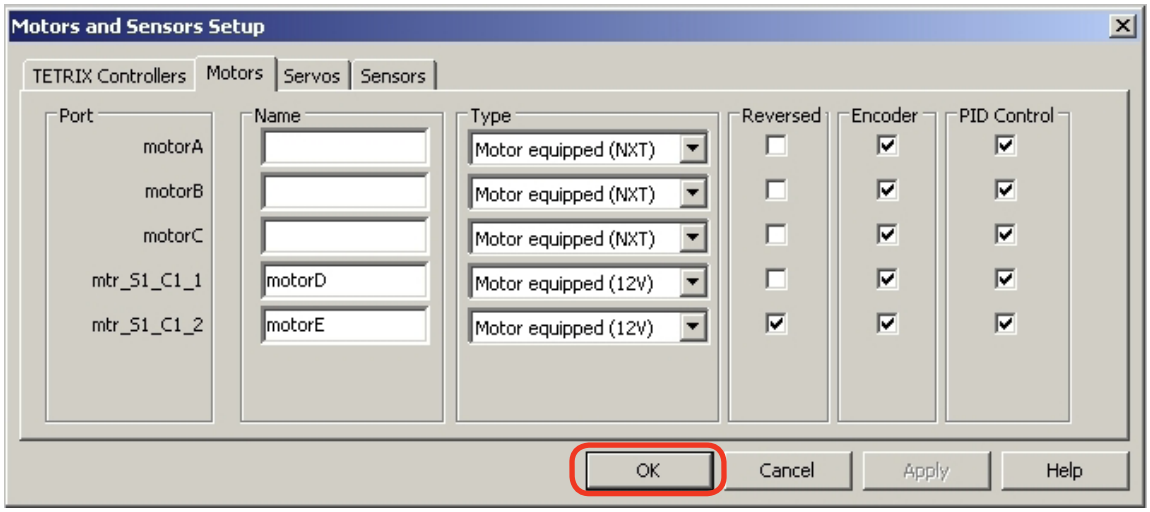


Now click the PID checkboxes for motorD and motorE to enable PID Control.

TETRIX™ Sensors

TETRIX Encoders (cont.)

Now that the encoders are configured and PID is enabled, click "OK" to save your settings and close the "Motors and Sensors Setup" window.



Once the window is closed, the pragma statements should be configured at the top of the page.

```
#pragma config(Hubs,  S1, HTMotor,  HTServo,  none,     none)
#pragma config(Motor,  mtr_S1_C1_1,     motorD,        tmotorNormal, PIDControl, encoder)
#pragma config(Motor,  mtr_S1_C1_2,     motorE,        tmotorNormal, PIDControl, reversed, encoder)
//*!!Code automatically generated by 'ROBOTC' configuration wizard               !!*//
```

```
task main()
{



}
```

Add a "task main" structure to your program, underneath the pragma statements.

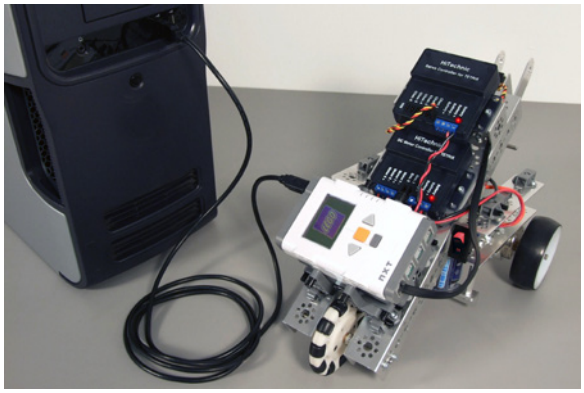
```
task main()
{
    motor[motorD]=40;
    motor[motorE]=40;
    wait1MSec(4000);
}
```

In the curly braces of the "task main" structure add this code:

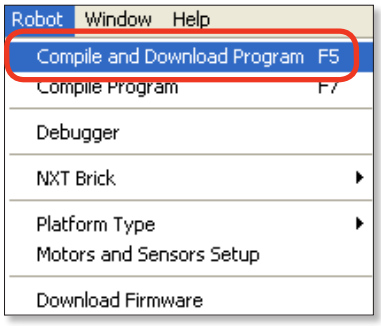
```
motor[motorD] = 40;
motor[motorE] = 40;
wait1MSec(4000);
```

This code will allow both motors to run in the forward direction at 40% power for four seconds.

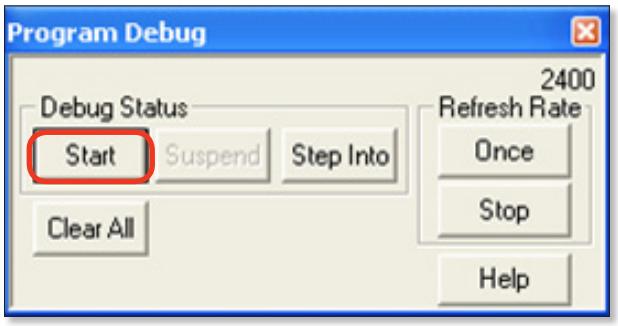
TETRIX Encoders (cont.)



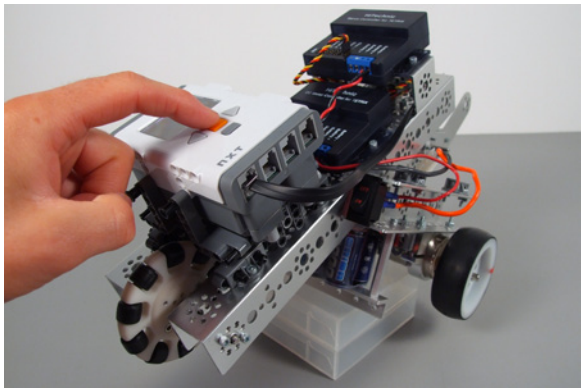
Make sure that the NXT and TETRIX Controllers are turned on, and that the USB cable is connected.



1. Go to the "Robot" menu.
2. Select "Compile and Download Program"



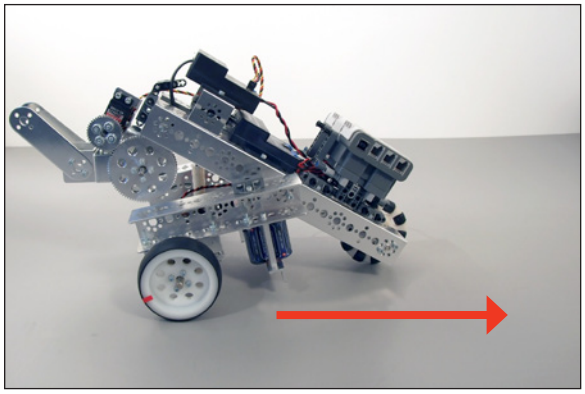
After the program downloads, a debugger window will appear. Prop the robot up on a table in a way that the wheels are not touching the surface. Use the debugger window to start your program once the robot is propped up.



While the robot is running the motors behave and sound differently than before. This is because the PID algorithm completes hundreds of updates per second, making minute, incremental changes to the motor speeds to compensate for outside forces like friction and resistance from gears.

TETRIX Encoders (cont.)

Unlike the NXT, you will not be able to see these updates in your NXT Devices debugger window. This is because the HiTechnic Motor Controller does the PID algorithm processing, instead of the NXT.



Place the robot on a flat surface and run it again. The robot should drive in a straighter line than it had without PID Control, because the PID algorithm checks and adjusts the motor speeds in order to keep the two motors closer to equal.

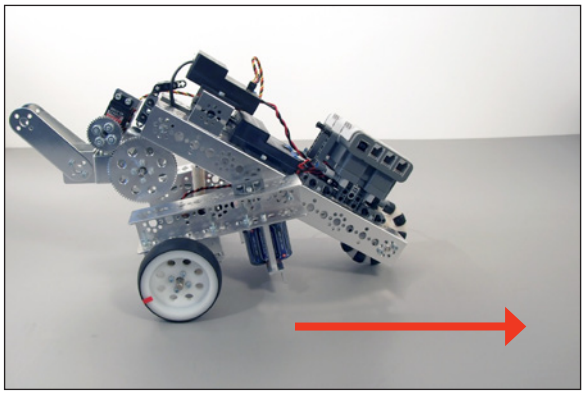
TETRIX nMotorEncoder

In this lesson you will learn how to use ROBOTC functions to make your robot travel a specific distance.

```
#pragma config(Hubs,  S1, HTMotor,  HTServo,  none,     none)
#pragma config(Motor,  mtr_S1_C1_1,     motorD,        tmotorNormal, PIDControl, encoder)
#pragma config(Motor,  mtr_S1_C1_2,     motorE,        tmotorNormal, PIDControl, reversed, encoder)
//*!!Code automatically generated by 'ROBOTC' configuration wizard               !!*//

task main()
{
  motor[motorD] = 40;
  motor[motorE] = 40;
  wait1Msec(3000);
}
```

Due to PID control, the robot will now move forward in a straight line for a specific amount of time.

However, the robot running forward for a specific amount of time is not always the best way to program the robot because most situations require the robot to travel a specific distance, not a specific time.



A ROBOTC function can help with the task of making the robot run for a certain distance. The function, “nMotorEncoder”, will give the robot feedback from its encoders as a value to use in the program.

Using the “nMotorEncoder” function, along with a looping structure, such as a “while loop”, allows the robot to travel forward until a certain distance is reached.

In the NXT-only Wall Detection lesson, the robot with a touch sensor moved forward until it touched a wall. This program will be very similar, but instead of waiting for the robot to touch a wall, the robot will move forward until it reaches a specific encoder count.

TETRIX nMotorEncoder (cont.)

```
task main()
{
  while(nMotorEncoder[motorD]>2880 ||
  nMotorEncoder[motorE]>2880)
  {
   motor[motorD]=40;
   motor[motorE]=40;
   wait1MSec(4000);
  }
}
```

In the "TETRIX Encoders" program, above the motor commands, add this code:

While(nMotorEncoder[motorD]> 2880 || nMotorEncoder[motorE] > 2880)

Add a set of curly brackets around the rest of the code.

The new line of code reads: "While the motor encoders attached to motorD is less than 2880 rotation counts OR the motor encoder attached to motorE is less than 2880 rotation counts."

Why 2880 encoder counts?

TETRIX encoders are more precise than the NXT encoders. The NXT encoder returns 360 encoder counts per revolution, which gives it accuracy to a single degree. However the TETRIX encoders return 1440 encoder counts per revolution, giving it accuracy to one quarter of a degree.

In order to make the robot travel two wheel rotations forward, multiply the counts per revolution:

1440 x 2 = 2880

Choosing an "OR" statement made sure that the robot waited until both motors were past the encoder count to ensure that both motors traveled the full distance

```
while(nMotorEncoder[motorD]>2880 ||
nMotorEncoder[motorE]>2880)
```

TETRIX nMotorEncoder (cont.)

```
task main()
{
  while(nMotorEncoder[motorD]>2880 ||
  nMotorEncoder[motorE]>2880)
  {
   motor[motorD]=40;
   motor[motorE]=40;
  }
}
```

Inside of the “while loop”, the program is going to have the motors more forward at 40% power.

Once the target is reached and the robot breaks out of the loop, the motors should stop. Remove the “wait1MSec” command from the program, otherwise the motors may not stop at the desired destination.

```
nMotorEncoder[motorD]=0;
nMotorEncoder[motorE]=0;
```

In order to make the encoders work properly, they must be reset whenever they are going to be used. This ensures that the encoders start at zero and do not begin counting from a previously stored value.

To reset the motor encoders, above the while loop add these lines of code:

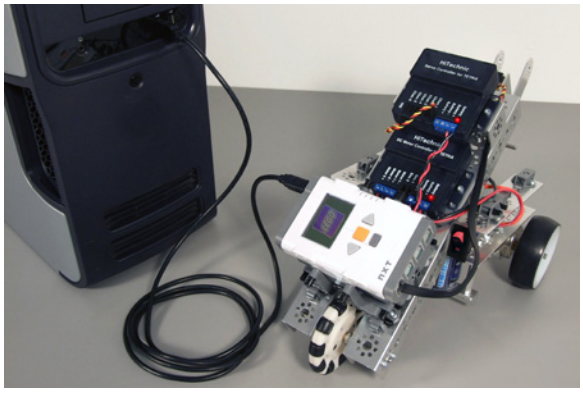
nMotorEncoder[motorD]=0;
nMotorEncoder[motorE]=0;

With three lines at the top of the program the encoders will be reset every time the program is run.

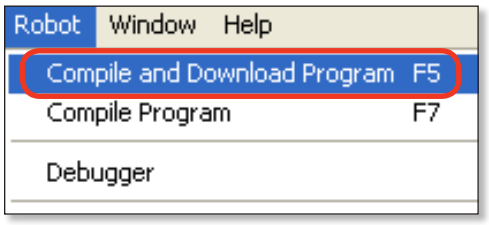
This is what the program should look like when it is complete:

```
task main()
{
  nMotorEncoder[motorD]=0;
  nMotorEncoder[motorE]=0;

  while(nMotorEncoder[motorD]>2880 ||
  nMotorEncoder[motorE]>2880)
  {
   motor[motorD]=40;
   motor[motorE]=40;
  }
}
```

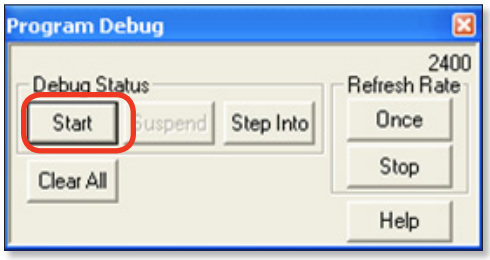
TETRIX nMotorEncoder (cont.)



Make sure that the NXT and TETRIX Controllers are both turned on, with the USB cable connected.

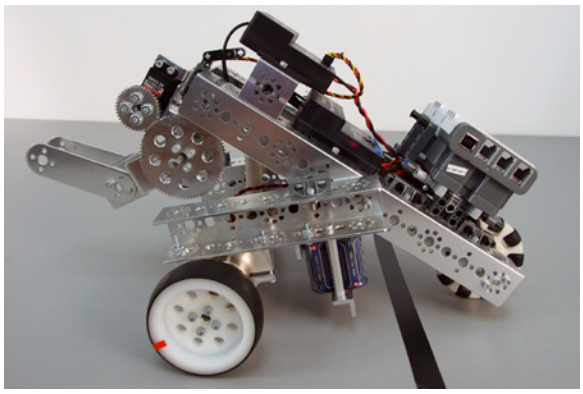


1. Go to the "Robot" menu.
2. Select "Compile and Download Program"



Once the program is finished downloading, the debugger window will appear.

Set the robot on a flat surface with enough room to run the program and then click the "Start" button.

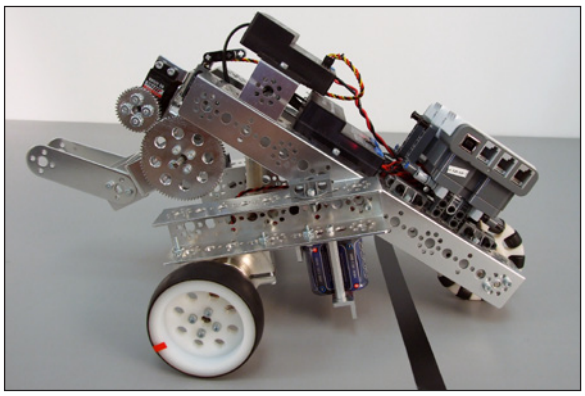


The robot should now travel forward for just about two rotation counts. However, you may notice that it is not as accurate as it should be. The delay between shutting down the motors and also the robot's momentum will cause it to continue moving forward slightly longer than expected.

Although this method is useful for accessing the values of the encoders and using them as feedback in your program, there is a better way to move to a specific encoder target destination. That approach will be explained in another lesson.

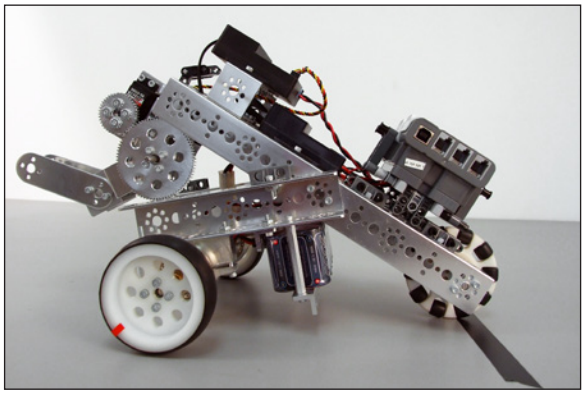
TETRIX nMotorEncoderTarget

In this lesson you will learn how to program the robot to travel to a specific distance and stop using two new functions: "nMotorEncoderTarget" and nMotorRunState.



"nMotorEncoderTarget" works differently than reading the encoder value with "nMotorEncoder".

With "nMotorEncoderTarget", ROBOTC is now doing the work to make sure that the robot reaches its target and stops at that target, rather than the less accurate method using a while loop, where the robot would only begin to stop once it reached its target.



"nMotorEncoderTarget" works by using a built in function of ROBOTC to control movement so that the robot travels a specific distance and stops.

This is automatically done by slowing the speed of the motors down as the robot gets closer to its target destination.

nMotorRunState:
Another function, called "nMotorRunState" will allow the robot to know if the motor is "Running", "Holding", or "Idle". Using "nMotorRunState" will allow the robot to know when it reaches its target. More on this later.

TETRIX nMotorEncoderTarget (cont.)

Example

Think of how different drivers operate their automobiles when approaching a Stop Sign.

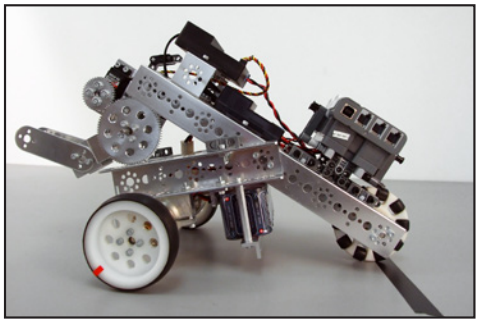
The first driver slams the brake pedal at the last minute, and though the car eventually stops, it can easily overshoot into the intersection.

The second driver begins to reduce acceleration as they approach the Stop Sign, easing the car forward as it slows, and usually ends up completely stopped directly next to the sign.

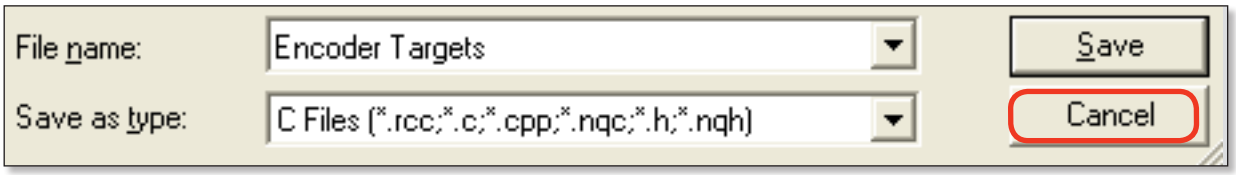
Application

1. Clear Encoders: Before the robot starts moving, the encoders must be reset.
2. Set Target: Tell the robot how far to travel by specifying a target "distance in rotation counts".
3. Turn on Motors: Next the motors have to be turned on in order to move forward.
4. Create Monitor: Lastly the robot has to know to wait until it has completed moving to the target distance.

The combination of these steps will cause the robot to stop at the line.



1. Open "nMotorEncoder" program.
2. Go to the "File" menu.
3. Click "Save As".
4. Save the program under the name "Encoder Targets".



TETRIX nMotorEncoderTarget (cont.)

Clear Encoders:
The encoders have been reset so that they start counting from zero.

```
task main()
{
  nMotorEncoder[motorD]=0;
  nMotorEncoder[motorE]=0;
```

```
task main()
{
  nMotorEncoder[motorD]=0;
  nMotorEncoder[motorE]=0;

  nMotorEncoderTarget[motorD]=2880;
  nMotorEncoderTarget[motorE]=2880;
```

Below the reset encoders part of the code, add the following lines of code:

nMotorEncoderTarget[motorD] = 2880;
nMotorEncoderTarget[motorE] = 2880;

These lines read:
"Set the Encoder Target of motorD to 2880 encoder counts and set the Encoder Target of motorE to 2880 encoder counts".

These two lines of code are what activate the "Move to Target" function in the ROBOTC firmware.

Set Target:
TETRIX Encoders return 1440 counts per wheel rotation, so to move the robot 2 wheel rotations:

1440 x 2 = 2880 encoder counts

```
task main()
{
  nMotorEncoder[motorD]=0;
  nMotorEncoder[motorE]=0;

  nMotorEncoderTarget[motorD]=2880;
  nMotorEncoderTarget[motorE]=2880;

   motor[motorD]=40;
   motor[motorE]=40;
```

We currently set the motor speed inside the while loop. However, we no longer need this while loop, so copy and paste the motor commands to be in between the "nMotorEncoderTarget" function and the while loop function.

The motor commands set the speed of the robot, but ROBOTC will override the programmed speed as the robot gets closer to its target to avoid overshooting the target.

Turn on Motors:
This step should be familiar to you by now. If not, go back and review the previous material.

TETRIX nMotorEncoderTarget (cont.)

Monitor the Robot's Run State:
A function, called "nMotorRunState" will allow the robot to know if the motor is "Running", "Holding", or "Idle". Using "nMotorRunState" will allow the robot to know when it reaches its target.

```
1. nMotorRunState[motorD]= runStateRunning;

2. nMotorRunState[motorD]= runStateHoldPosition;

3. nMotorRunState[motorD]= runStateIdle;
```

Although the while loop that was in the program before was clearly unnecessary, using the "MotorRunState" to create a while loop is needed to make the robot wait for completion of the "nMotorEncoderTarget" function. However the robot should not execute any other code while waiting for the movement to complete. Therefore it is necessary to create an "Idle Loop" to have the program wait until the motors come to a stop.

Modify the existing while loop with the following code

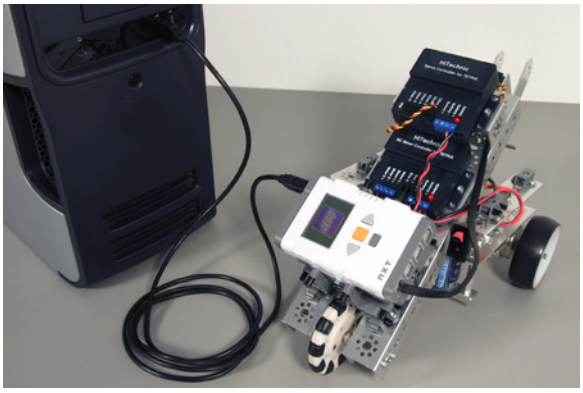
while(nMotorRunState[motorD] != runStateIdle || nMotorRunState[motorE] != runStateIdle)

```
task main()
{
  nMotorEncoder[motorD] = 0;
  nMotorEncoder[motorE] = 0;

  nMotorEncoderTarget[motorD] = 2880;
  nMotorEncoderTarget[motorE] = 2880;

  motor[motorD] = 40;
  motor[motorE] = 40;

  while(nMotorRunState[motorD] != runStateIdle || nMotorRunState[motorD] != runStateIdle)
  {

  }
}
```

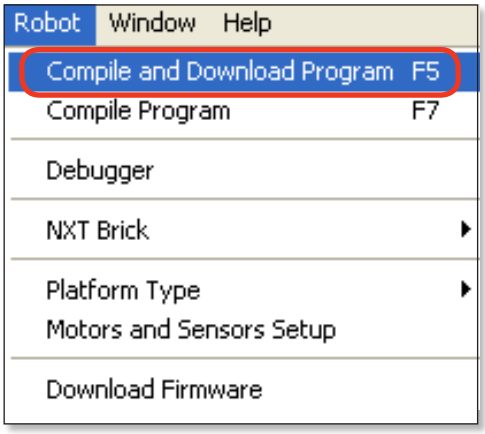
The code above reads:
"While the run state of motorD is NOT equal to the state "idle" OR the "run state" of motorE is NOT equal to run state "idle"."

The robot will stay inside of the while loop until both motors reach the "idle" state, which means they are stopped.

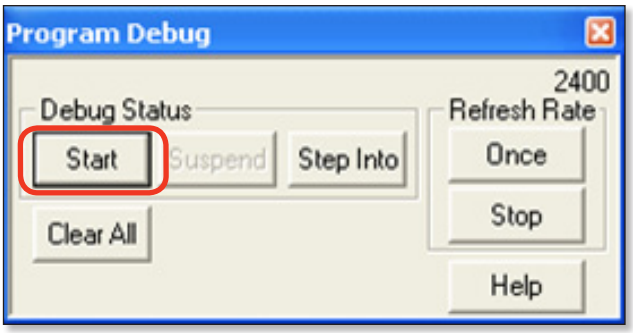
TETRIX nMotorEncoderTarget (cont.)



Make sure that the NXT and the TETRIX Controller are both turned on, with the USB cable connected.



1. Go to the “Robot” menu.
2. Select “Compile and Download Program”

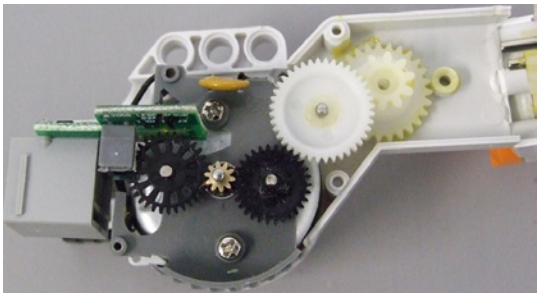


Once the program is finished downloading, the debugger window will appear.

Set the robot on a flat surface with enough room to run the program and then click the “Start” button. The robot should run forward for two wheel rotations and stop.

LEGO Encoders

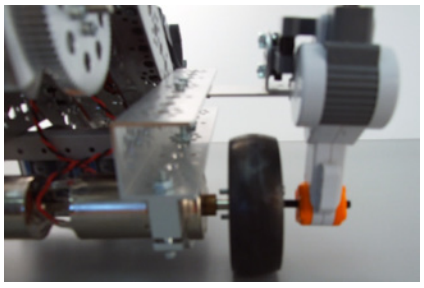
In this lesson you will learn how to use feedback from the sensors built into the NXT motors to control the TETRIX DC motors.



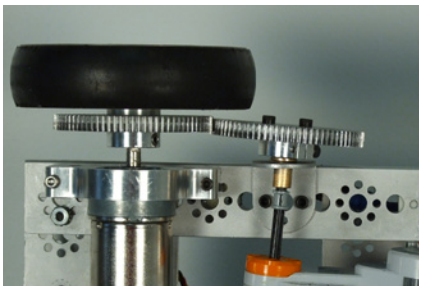
The NXT motors have encoders built into them. Encoders are sensors that act as counters and count the number of revolutions that the wheel turns.

When properly engineered, the built in encoders from the NXT motors can give feedback to the TETRIX DC motors, enabling accurate control.

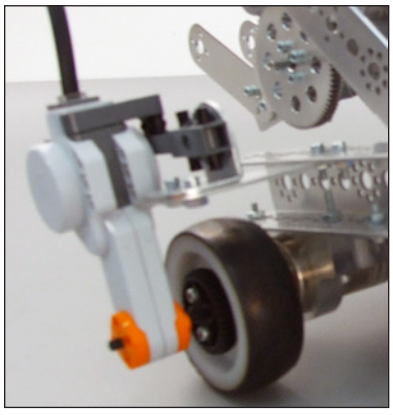
In order to use the NXT motor's built in encoder, the NXT and TETRIX motors must be connected to the same axle. This can be done by using a direct connection or a series of gears. The key is that the TETRIX motor drives the NXT motor.



Direct Connection



Gear Connection



The "nMotorEncoder" function enables the robot to access the value of the encoders built into the NXT motors in real time.

There are three steps for using the "nMotorEncoder" function and NXT motors to control the TETRIX motors:

1. Name the encoder.
2. Reset the encoder to zero
3. Place the value of "nMotorEncoder" into a conditional statement that controls the TETRIX motors.

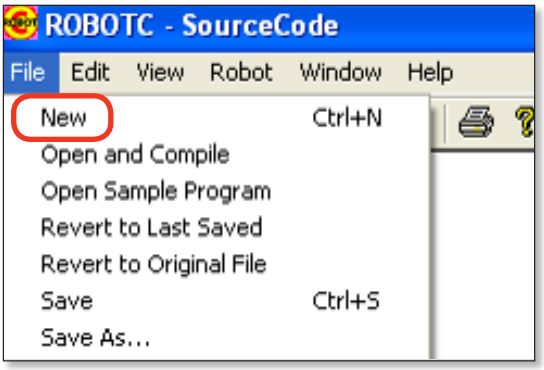
How to do this will be taught in the next lesson.

Before moving onto the next lesson, make sure that your NXT motors are attached correctly to your robot.

LEGO Encoders Part 2

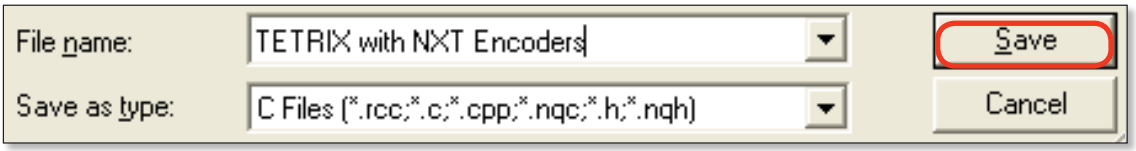
What are we programming?

The goal is to program the TETRIX robot to travel three wheel rotations forward and then stop.

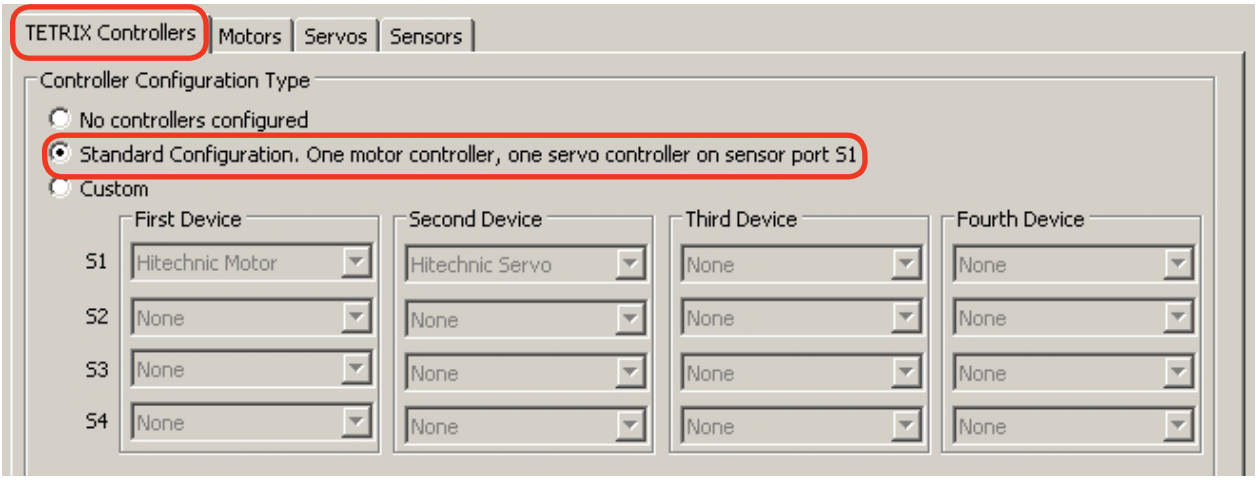


1. Go to the "File" menu.
2. Select "New".

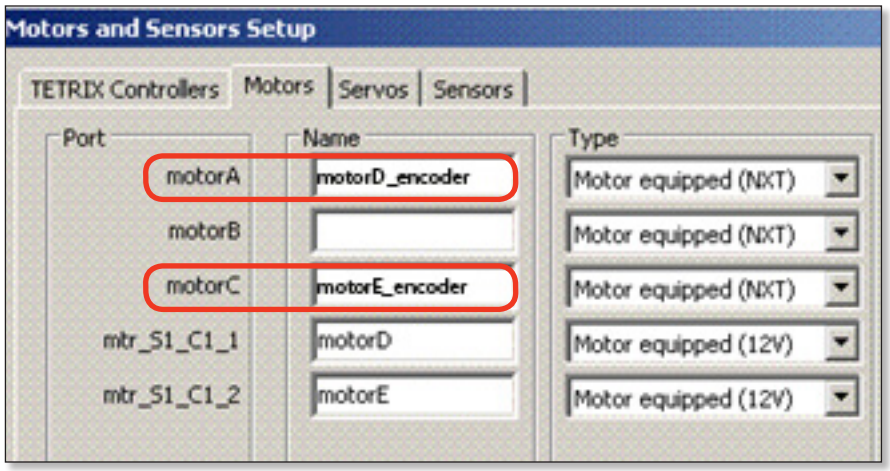
3. Go to the "File" menu.
4. Select "Save As".
5. Save program as 'TETRIX with NXT Encoders".



6. Go to the "Robot" menu.
7. Select "Motors and Sensors Setup"
8. Click on the "TETRIX Controllers" tab
9. Select "Standard" Configuration.

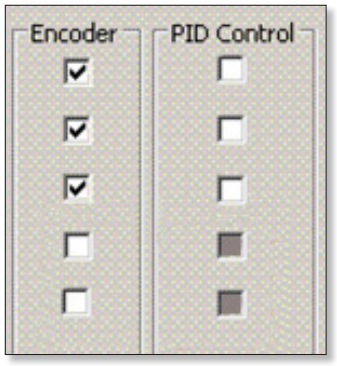


LEGO Encoders (cont.)



In the "Motors" tab, rename the NXT motors something that is more intuitive:

motorA = motorD_encoder
motorC = motorE_encoder



PID will not work with this configuration, therefore the PID function must be unchecked for the NXT motors.

Click "OK" to close the window.

```
task main()
{



}
```

Add a "task main" structure to the program.

LEGO Encoders (cont.)

Remember:
motorA has been renamed motorD_encoder. Therefore, to see the encoder value of motorA the program would read “nMotorEncoder[motorD_encoder]”. The name of motorC has also been changed.

```
nMotorEncoder[motorD_encoder]

nMotorEncoder[motorE_encoder]
```

Anytime the encoders are used, they must be reset.

Reset the encoders by adding:

nMotorEncoder[motorD_encoder] = 0;
nMotorEncoder[motorE_encoder] = 0;

```
task main()
{
  nMotorEncoder[motorD_encoder]=0;
  nMotorEncoder{motorE_encoder]=0;


}
```

The next step is to setup a condition that will check to see that the robot has not traveled too far.
The goal is to travel three rotations and stop; this must be translated into language that the robot understands.

(To review while loops and conditional statements, go back to the Wall Detection Touch section of the curriculum.)

Remember:
The encoders inside of the NXT motors count to 360 for every wheel rotation.
To get the number of degrees for a set of rotations, multiply the number of rotations by 360.

3 x 360 = 1080

LEGO Encoders (cont.)

A while loop can constantly check both encoders to make sure that the robot has not reached, nor gone beyond, its target distance.

The "nMotorEncoder" function can be used to read and access the encoder values.
The robot will be reading the value and comparing it to a predetermined distance.

Add the following code:

while(nMotorEncoder[motorD_encoder] < 1080 || nMotorEncoder[motorE_encoder] < 1080)

Don't forget the curly braces to enclose the while loop structure.

```
task main()
{
  nMotorEncoder[motorD] = 0;
  nMotorEncoder[motorE] = 0;

  while(nMotorEncoder[motorD] < 1080 || nMotorEncoder[motorE] < 1080)
  {

  }
}
```

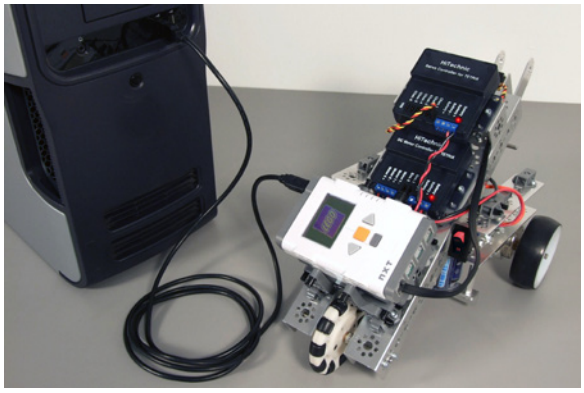
Inside of the loop, the DC motors should be traveling forward at 25% speed.
Add these lines of code:

motor[motorD]=25;
motor[motorE]=25;

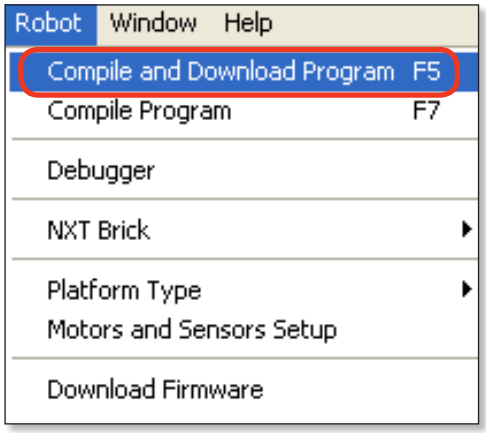
Outside of the loop, no programming is necessary, because the robot just needs to stop once it reaches its destination.

```
task main()
{
  nMotorEncoder[motorD] = 0;
  nMotorEncoder[motorE] = 0;

  while(nMotorEncoder[motorD] < 1080 || nMotorEncoder[motorE] < 1080)
  {
    motor[motorD] = 25;
    motor[motorE] = 25;
  }
}
```

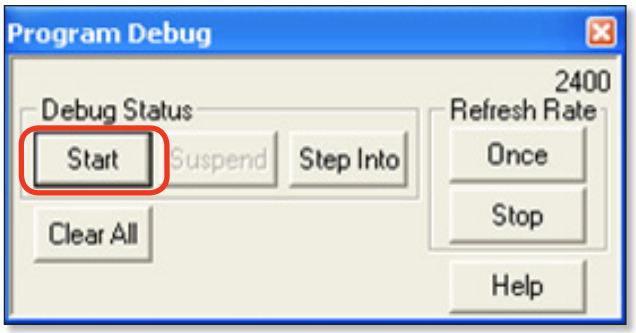
LEGO Encoders (cont.)



Make sure that the NXT and the TETRIX Controller are both turned on, with the USB cable connected.



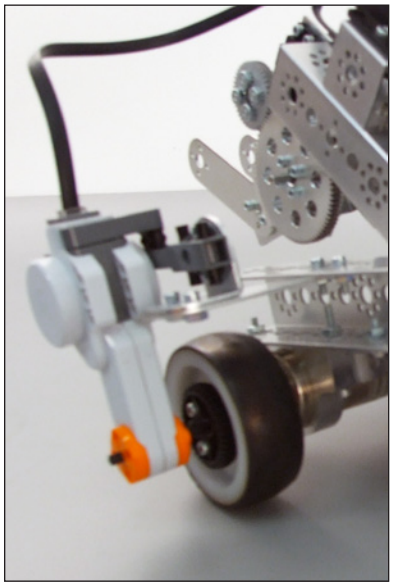
1. Go to the "Robot" menu.
2. Select "Compile and Download Program"



Once the program is finished downloading, the debugger window will appear.

Set the robot on a flat surface with enough room to run the program and then click the "Start" button. The robot should run forward for about three wheel rotations and stop.

LEGO Encoders (cont.)



There are some things to keep in mind while using this method to determine the distance the robot has traveled. Because the encoders being used were not designed specifically for the TETRIX system, some functionally is lost.

Both PID control and the “nMotorEncoderTarget” function are not available when using the NXT motors with the TETRIX DC Motors in this manner. The robot is only able to access the raw encoder values from the NXT motors. Accuracy can also be compromised if any of the components are out of position or not properly secured.