Introduction to RobotC and Tetrix with Lego Mindstorms

These lessons will introduce writing autonomous robot programs in RobotC using the Tetrix Building system and the Lego Mindstorms Robotics Kit. Topics will cover:

- Selecting the Type of Robotics Platform
- Configuring the Motors, Servos, and Sensors using the RobotC Setup Option.
- Basic Forward and Reverse Program using Motor Blocks and Wait commands.
- How to create methods (reusable sections of Code)
- Creating Methods with Sensors and using While Loops:
    - TouchStop with Touch Sensor
    - NearStop with UltraSonic Sensor
    - Dark Stop with Light Sensor
- Using the Timer
- More advanced methods: (Application)
    - Minimum Distance Seeker
    - Proportional Wall Following with UltraSonic
    - Proportional Line Following

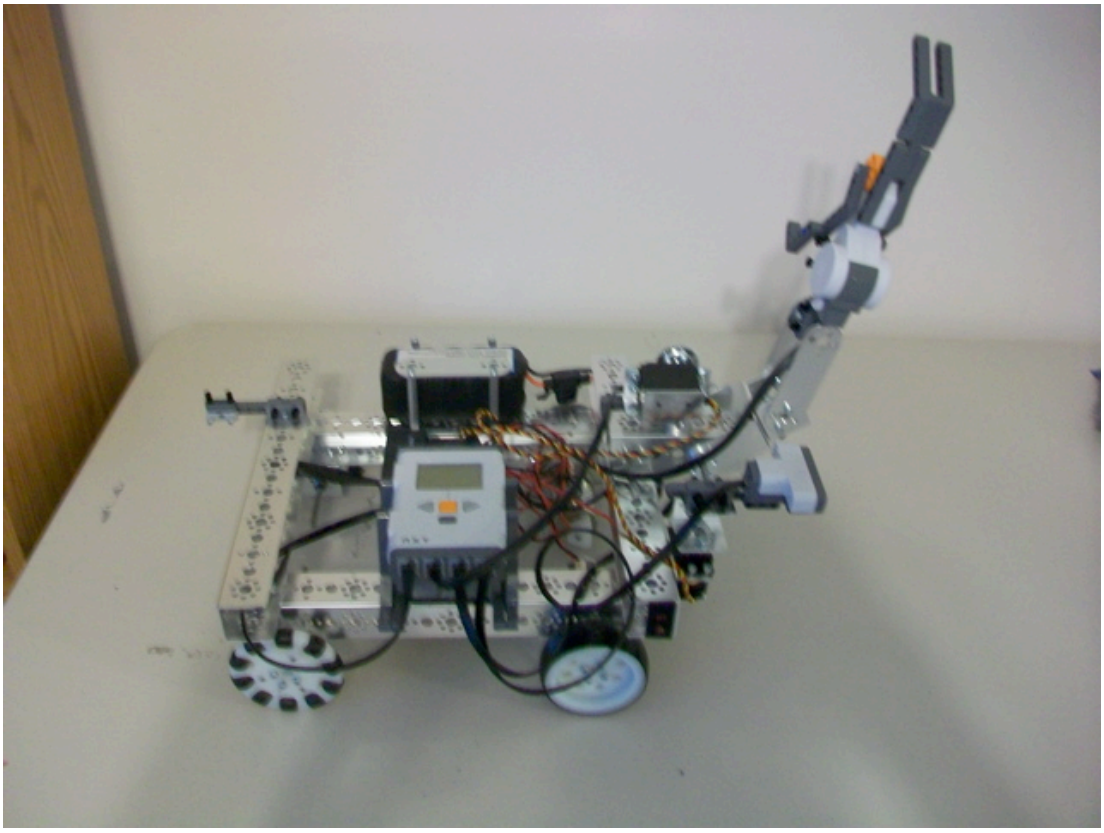
Lesson 1: Setting up the Robot Platform and Configuring the Motors, Servos, and Sensors.

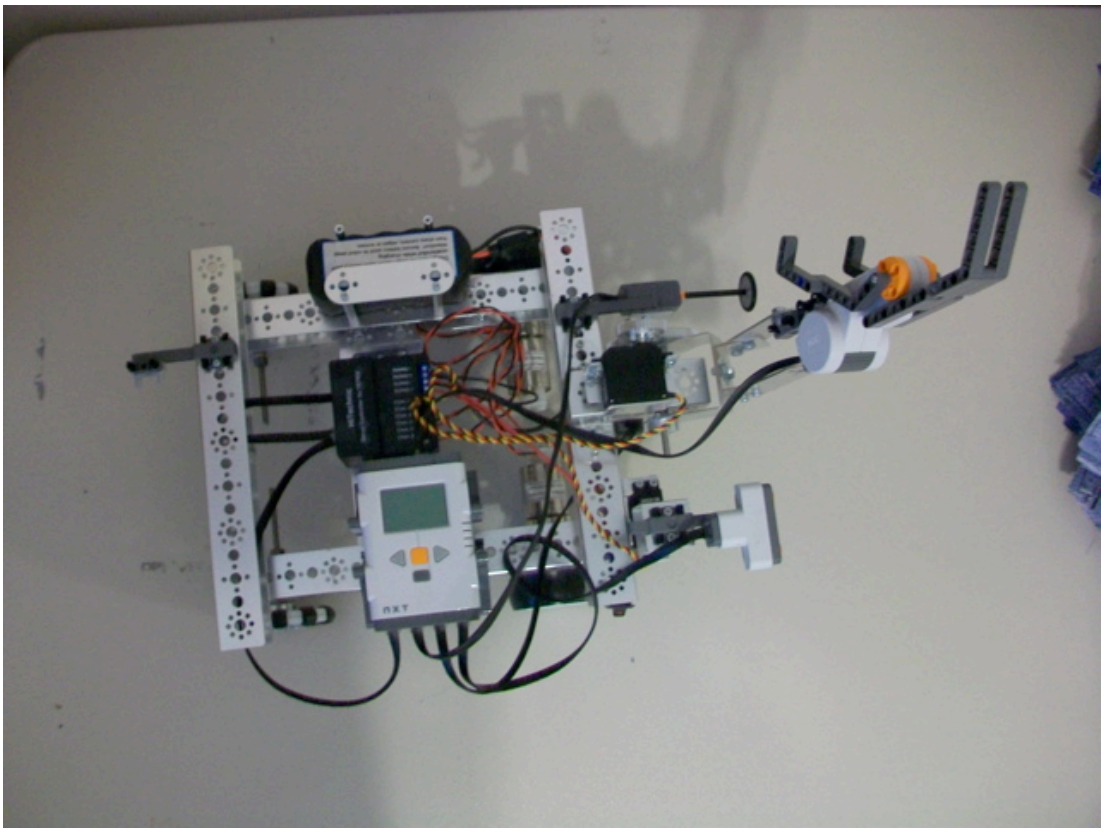
This tutorial uses the "ICEBot" frame with the left and right drive wheels being on the front of the robot. The NXT Brick is configured as follows:

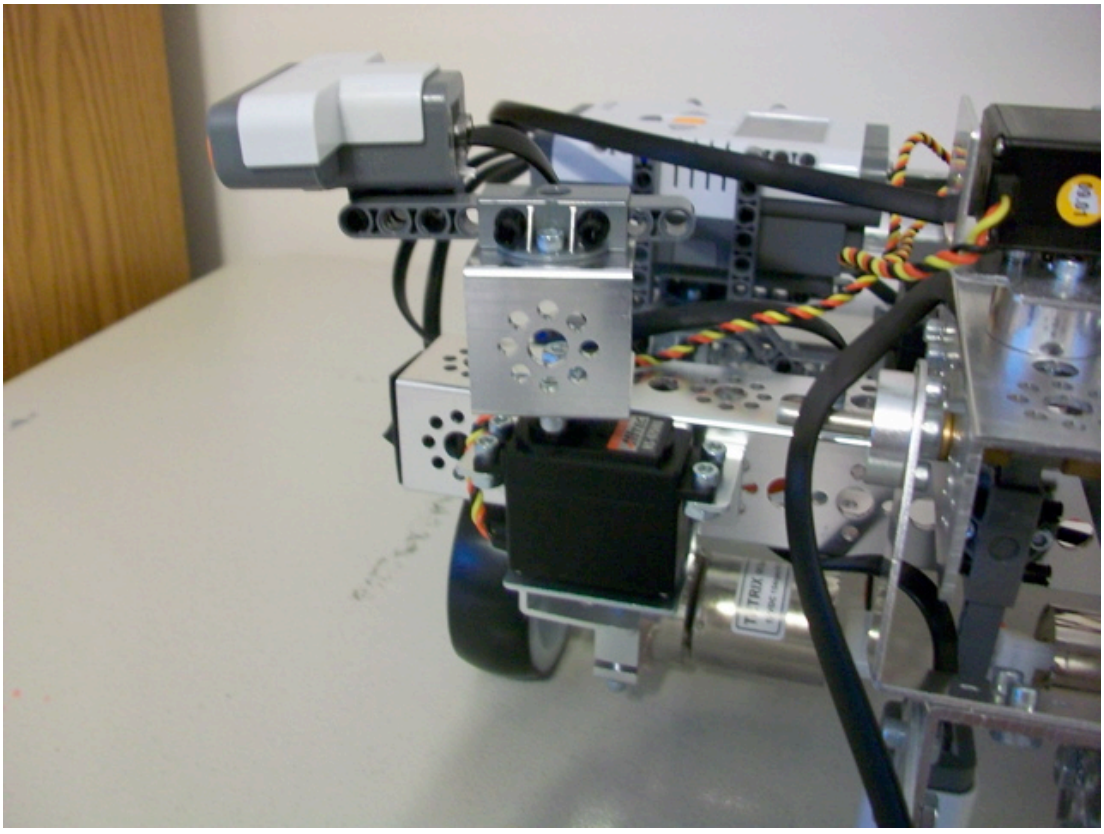
Sensor Ports:
1 = Motor and Servo Controller for the Tetrix motors and servos
2 = NXT Touch Sensor
3 = NXT Light Sensor
4 = NXT UltraSonic Sensor

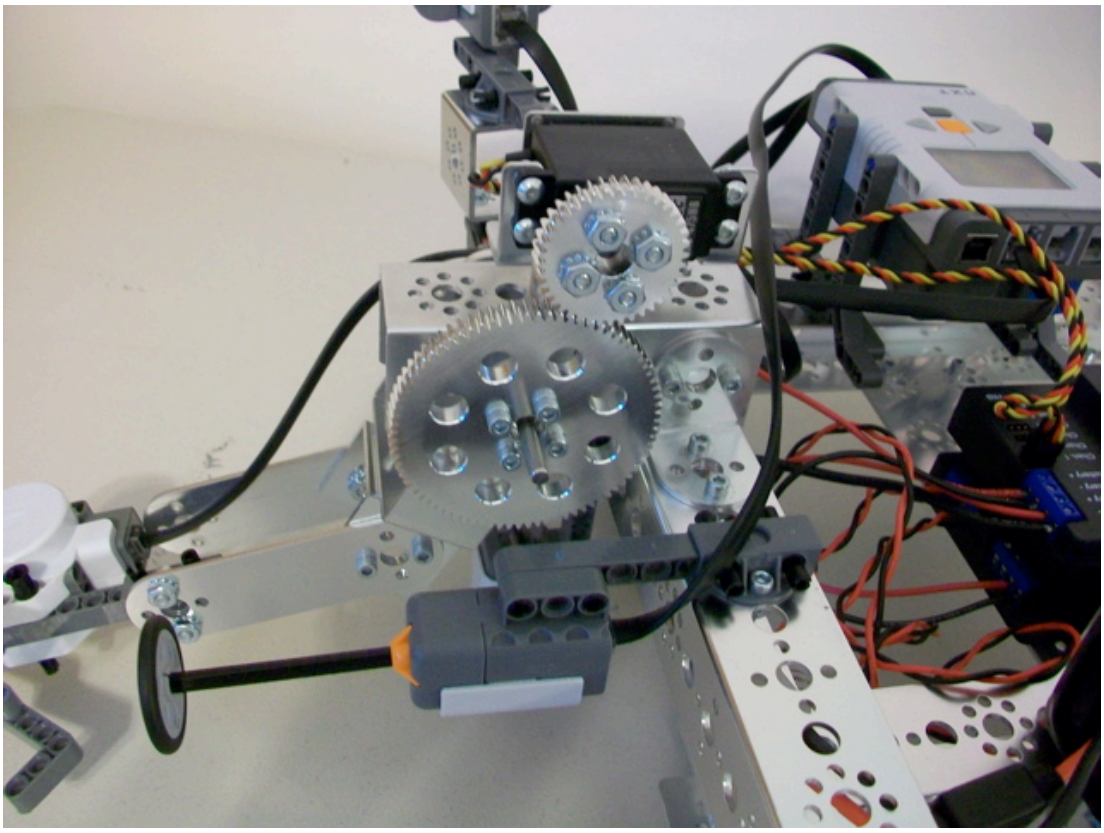
Motor Ports:
C = NXT Motor to control the "Hand"

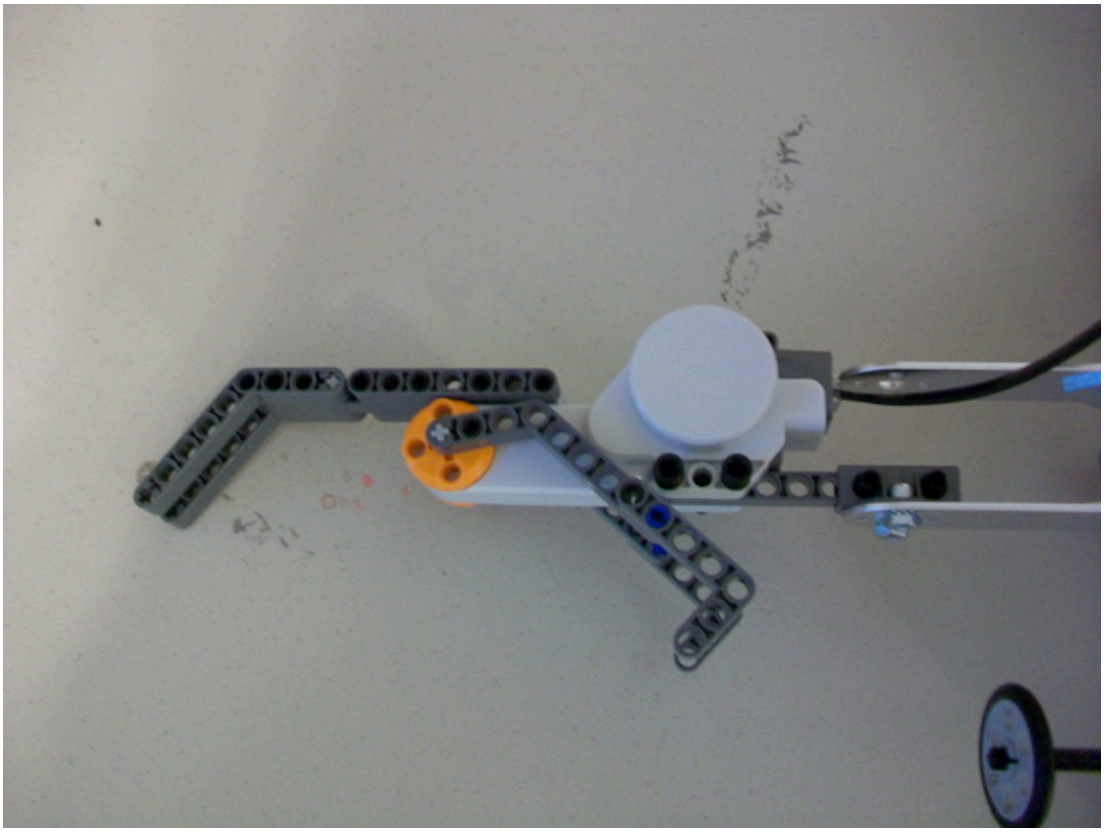
ICEBot with Michaud Modifications Images:

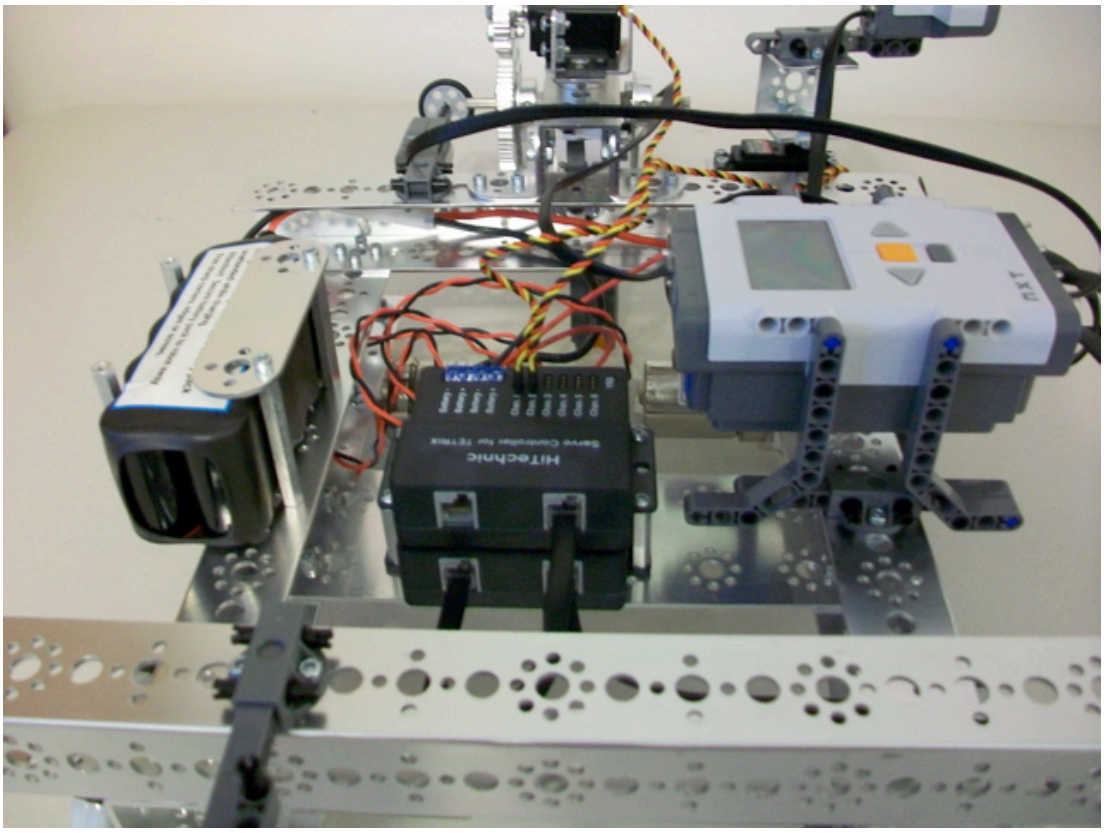


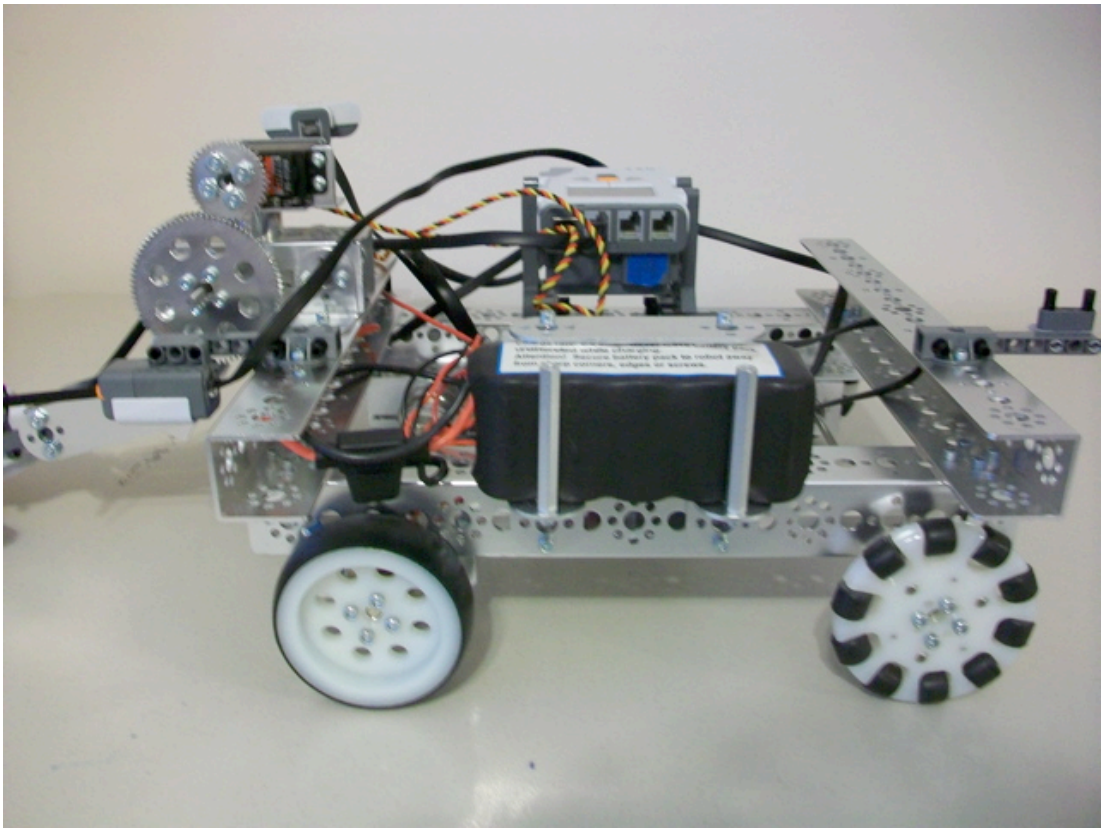

NXT







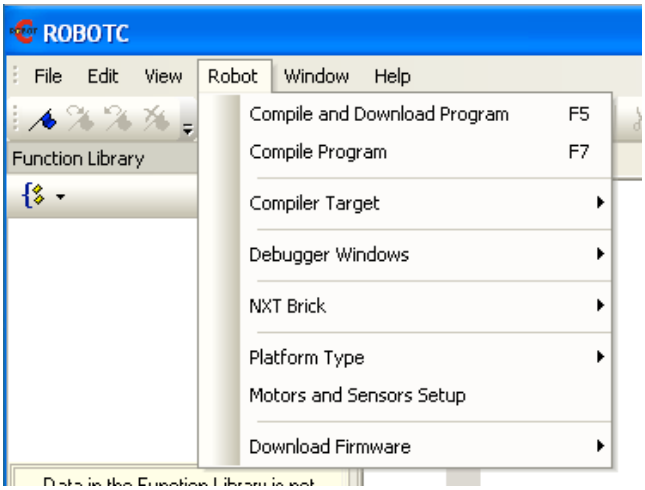

HiTechnic
NXT



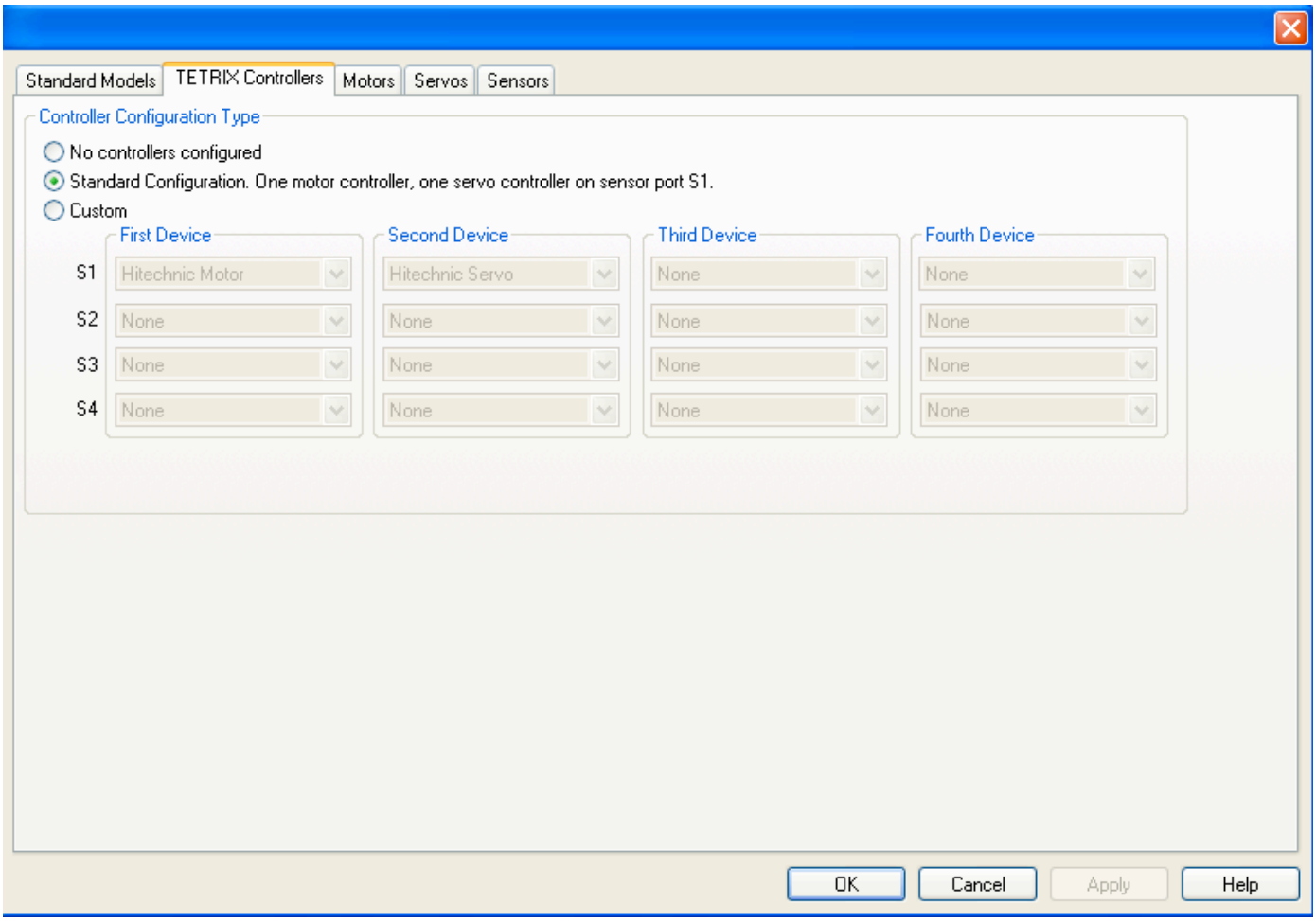
When you start programming in RobotC, you will need to define all the motor, servo, and sensor connections. This is like defining the "objects" in a program before you start creating instructions and methods. RobotC has two menu driven tools to accomplish this configuration.

Steps:

1. Start RobotC and Select File-New to start a new program.
2. Select "Robot->Platform Type" from the menu bar and select "NXT+Tetrix".
3. Select "Robot->Motors and Sensors Setup"

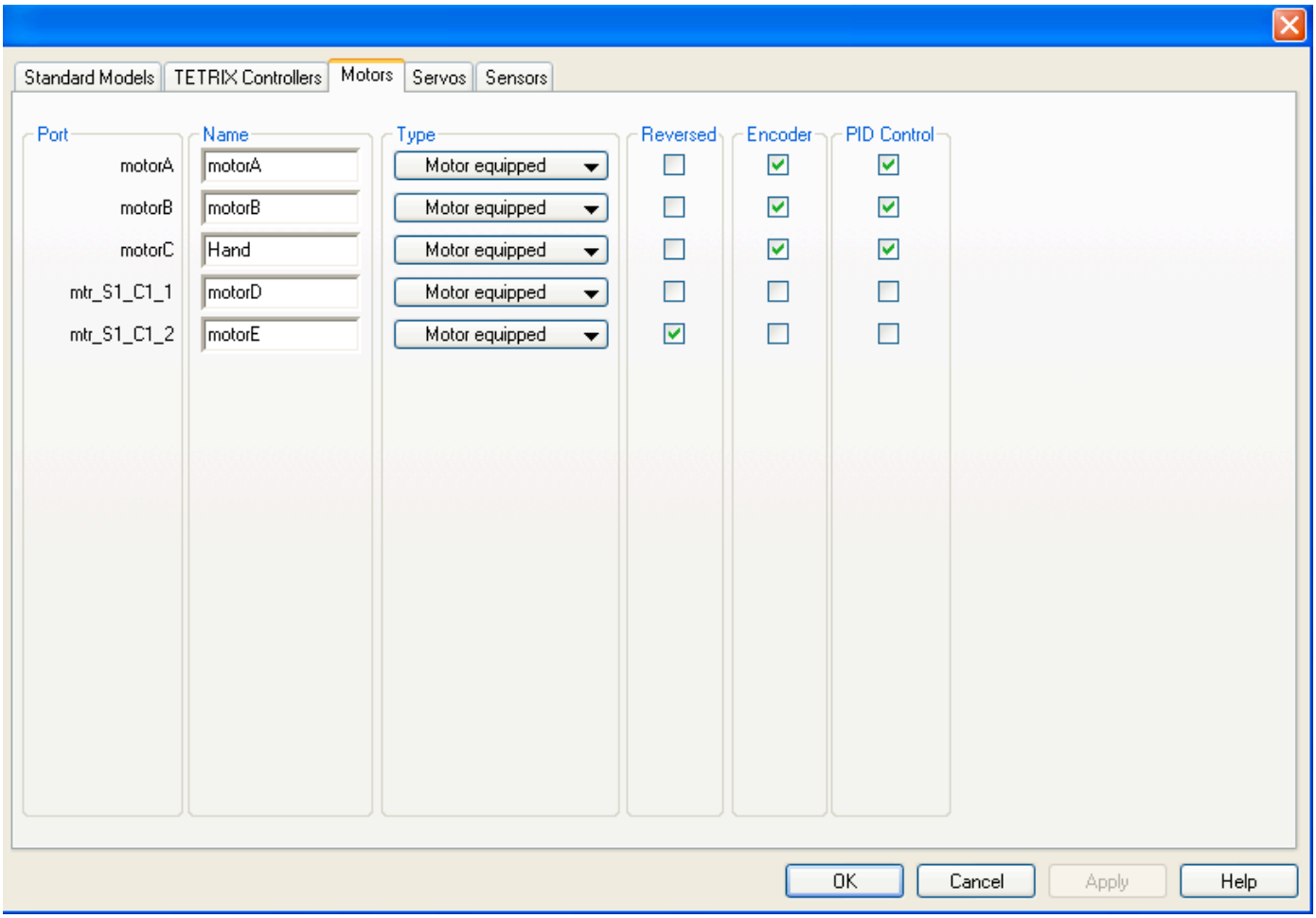


4. Select the "TETRIX Controllers" tab and select the second option: "Standard Configuration. One motor controller, one servo controller on sensor port S1."
(Though in more advanced robot designs, you can set up your own configuration)



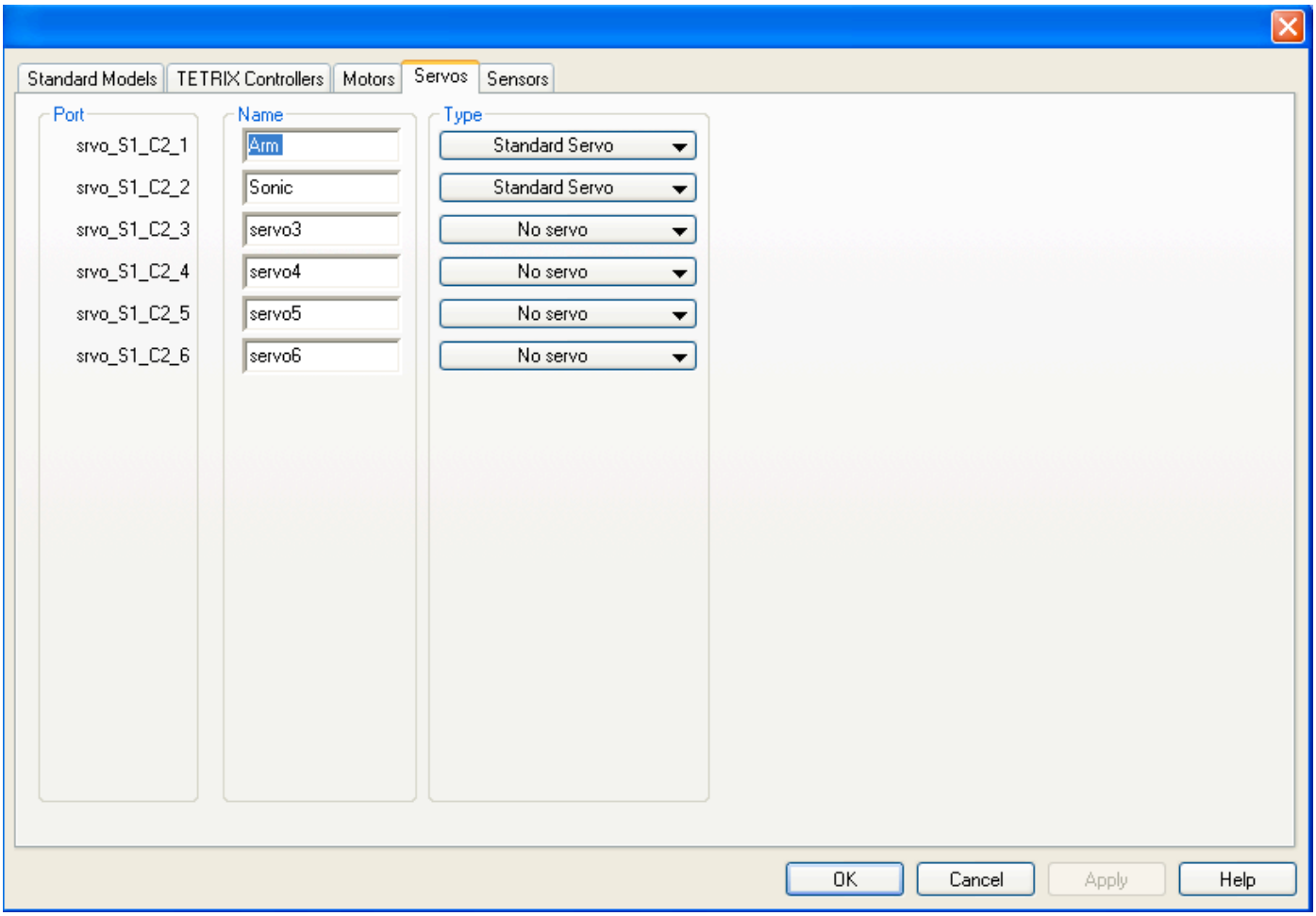
5. Click "Apply"

6. Click the "Motors" tab. Name the motors as shown below. Note that I used the name "Hand" for "motorC." By convention, motors "A, B, and C" are the NXT motors. Motors "D and E" are the Tetrix DC motors. (12V). Make sure you select "Reversed" for motorE. This allows motors to rotate "forward" and "backward" in vehicle type robots. Note that you can give the Motors names. Choose names that make sense for the robot's purpose. This will make your Code easier to understand later.



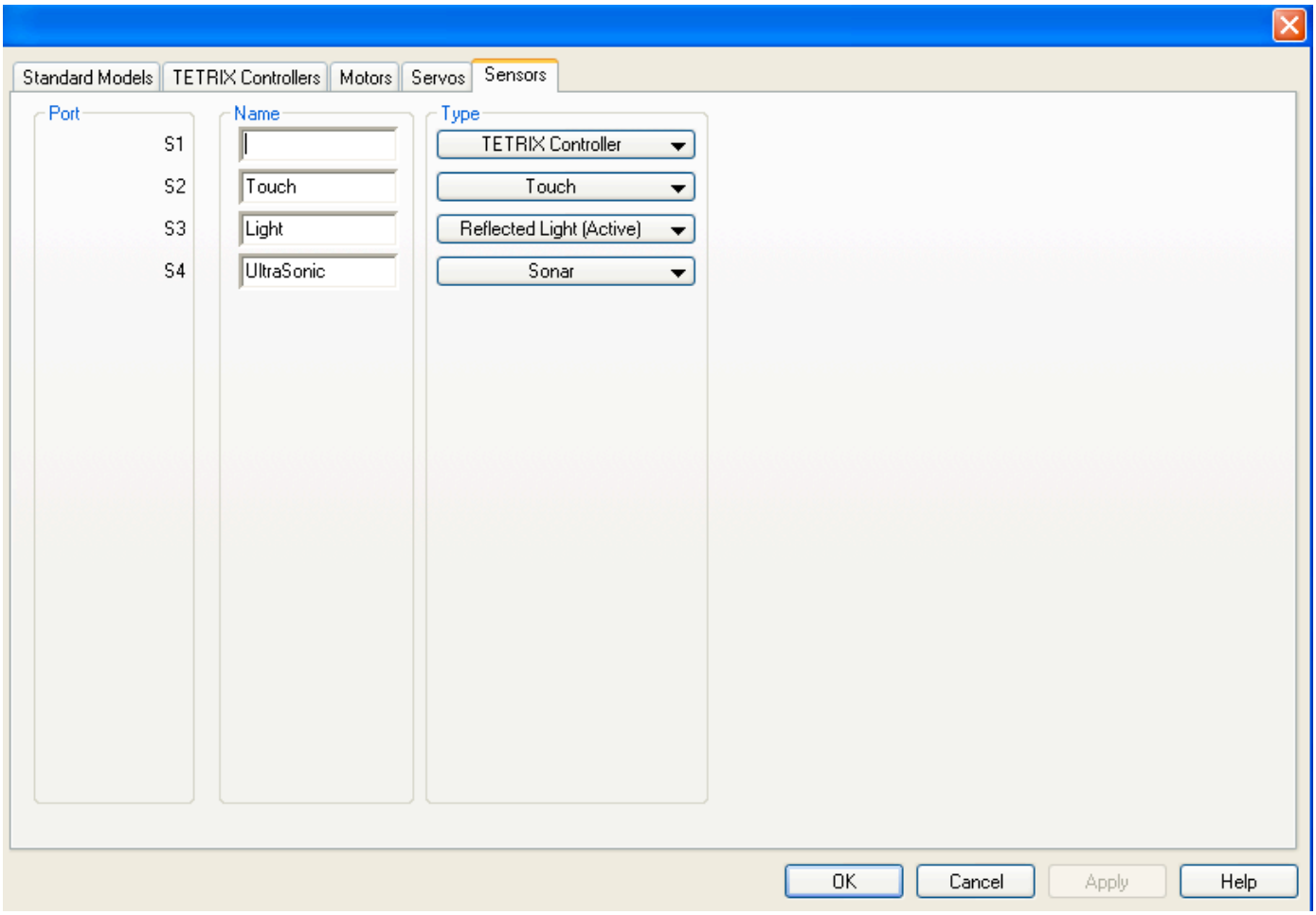
7. Click "Apply."

8. Click the "Servos" tab. Name servo1 "Arm" and select "Standard Servo". Note that I also have servo2 named "Sonic."



9. Click "Apply."

10. Click the "Sensors" tab. Set Sensors S2, S3, and S4 as shown below. Make sure to select "Touch", "Reflected Light", and "Sonar." in "Type."



11. Click "Apply."

12. Click "OK."

13. Note that the first 10 lines of your program now contain auto generated code that configures the Hubs, Sensors, Motors, and Servos. (The "pragma config") These define the robot objects in the program.

14. Save your program as “Tetrix Train”

Lesson 2: Beginning to Drive: The "Forward - Stop"

The most basic element of robot programming follows the format of

"Start the Task -> Wait for a condition -> Stop the Task"

In this program, we will:
1. Start the motors
2. Wait for 2 seconds
3. Stop the motors

In your program - type the following code:

```
task main()
{
  motor[motorD] = 25;
  motor[motorE] = 25;
  wait1Msec(2000);
  motor[motorD] = 0;
  motor[motorE] = 0;
}
```

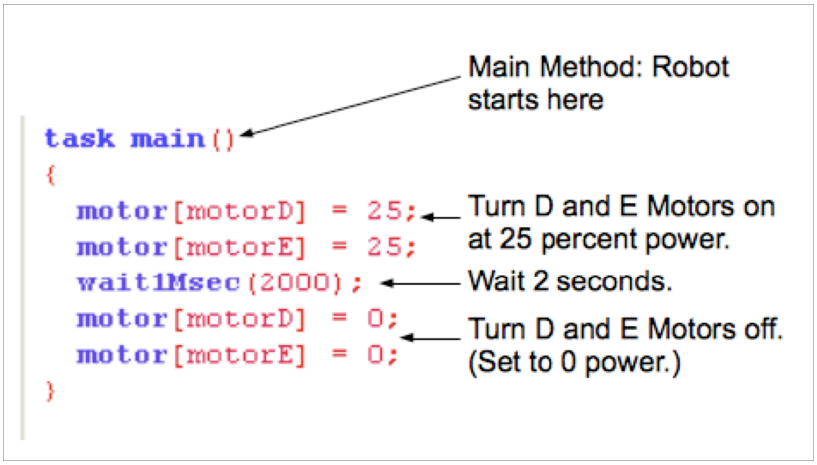
To run your program:
1. Save your Program (File->Save). Name it “Tetrix Training.”
2. Select "Robot -> Compile and Download Program." Then close the "Debugger" and select and Run the program on the NXT Brick.

A few notes of Caution:
1. Always run your robot on the floor or a walled table. Tetrix robots are heavy and can do some serious damage to you or other objects.

2. Keep your fingers away from the metal gears.

All RobotC programs start with the "task main." Inside the curly brackets are the instructions:

Here is a breakdown:



Now try the following program "Out and Back"

```
// Out and Back Program

task main()
{
  motor[motorD] = 25;
  motor[motorE] = 25;
  wait1Msec(2000);
  motor[motorD] = 0;
  motor[motorE] = 0;
  wait1Msec(2000);
  motor[motorD] = -25;
  motor[motorE] = -25;
  wait1Msec(2000);
  motor[motorD] = 0;
  motor[motorE] = 0;

}
```

Extras:

There are two types of Turns:

1. Swing Turn: One wheel rotates and the other stays still
2. Point Turn: One wheel turns forward while the other wheel turns backward.

How would you program the robot to do Swing and Point turns?

Lesson 3: Setting the Servo Positions

The Tetrix Servos act like motors that you can lock in positions ranging from 0 to 255. You set these positions like setting a variable.

Example: (Moves the Front Arm Up and Down with 1 second delays)

```
task main()
{
  servo[servo1] = 0;
  wait1Msec(1000);
  servo[servo1] = 155;
  wait1Msec(1000);
}
```

I suggest in each of your programs, put a "servo[servo1] = 0;" before each program to lift up your arm.

Lesson 4: Making methods

One of the main goals of programming is to create re-usable pieces of directions that make sense to the human reading your Code, and save time and space (not having to re-type long series of commands over and over again.) Thus we are going to start making a series of "Methods" that do common tasks.

A "Method" is a block of instructions that you can call in the Main Task over and over again. We will make two methods: "driveStraight" and "stopMotors."

Type the following above your "task main()":

```
// Methods
// Drive Straight

void driveStraight(int power)
{
  motor[motorD] = power;
  motor[motorE] = power;
}

// stopMotors
void stopMotors()
{
  motor[motorD] = 0;
  motor[motorE] = 0;
}
```

Note:
1. We use the term "void" before the Method because the method does not return a value. (It only directs action.)
2. The "int power" is a parameter for the driveStraight method. This allows the programmer to define how much power will go to the motors. The "int" means "integer."

Type the following into the task main() to run these methods:

```
// Drive Straight
void driveStraight(int power)
{
  motor[motorD] = power;
  motor[motorE] = power;
}

// stopMotors
void stopMotors()
{
  motor[motorD] = 0;
  motor[motorE] = 0;
}

task main()
{
  driveStraight(25);
  wait1Msec(2000);
  stopMotors();

}
```

Note that the main task now uses three lines of code to do what took us 5 lines earlier.

Exercises:

1. Make the Robot Drive Forward and Backward using the driveStraight and stopMotors methods. Hint: Negative numbers will make the motors turn backwards.

2. Create a Method for Turning the Robot Left and Right. Then program your Robot to Drive in a Square.

Lesson 5: Methods using Sensors and While Loops

Because Tetrix DC motors do not have encoders or rotation sensors, we will need to use other Sensors such as Touch, Light, and Sonar to help guide the robot through the world. These series of methods follow this basic robot pattern:

"While a Condition is True: Do an Action. When the Condition is not true: Stop the motors or Action"

Add the following method and edit the main task in your program for "TouchStop." Save, Compile and test the program: (Remember, you will need to keep the Methods “driveStraight” and “stopMotors” in your program.

```
// touch Stop method
void touchStop(int power)
{
  while (SensorValue[Touch] != 1)
  {
    driveStraight(power);
  }
  stopMotors();
}



task main()
{
  touchStop(25);

}
```

Notes:
1. We use the parameter "int power" again to let the programmer define how fast the robot should go.
2. The "while loop" tells the program to run the "driveStraight" method until the touch sensor is touched.
3. "SensorValue[Touch] != 1" means: "Touch Sensor does not equal 1" or "Touch Sensor Not Pressed."
4. Translation of method: "While the Touch Sensor is not pressed, Drive Straight. When the Touch sensor is pressed, stop the motors."

Try these methods for UltraSonic Stop and Light Stop:

```
// near Stop method
void nearStop(int distance, int power)
{
  while (SensorValue[UltraSonic] < distance)
  {
    driveStraight(power);
  }
  stopMotors();
}

task main()
{
  nearStop(20, 25);

}
```

```
// darkStop
void darkStop(int dark, int power)
{
  while (SensorValue[Light] > dark)
  {
    driveStraight(power);
  }
  stopMotors();
}

task main()
{
  darkStop(40, 25);

}
```

Notes:
1. In the "nearStop" method, note the two parameters: "distance" and "power." This allows the user to define the distance in centimeters for the stop and the speed of the robot.

2. In the "darkStop" program the parameters "dark" and "power" allow the user to specify the darkness the robot is looking for and the speed of the motors.

We can also reduce the basic Forward for Time type program with this method:

```
// forward for seconds method
void driveTime(int seconds, int power)
{
  driveStraight(power);
  wait1Msec(seconds * 1000);
  stopMotors();
}

task main()
{
  driveTime(2, 25);
}
```

Exercise:

1. Using the “nearStop” method, create a program that will allow your robot to drive straight and stop if it nears any obstacle. Then direct the robot to turn to a new direction and keep moving.

Lesson 6: More Advanced Programs - using variables and methods we have already created.

Example 1: The Nearest Object program. This example spins the robot around. While the robot is spinning, it is seeking out the nearest object using the UltraSonic Sensor. When it is finished spinning, it points back to the nearest object and travels up to the object and stops.

```
// Near Seek Method
void minDistanceSeek()
{
  int minimumDistance = (SensorValue[UltraSonic]);
  int currentDistance = 0;
  ClearTimer(T1);
  while (time1[T1] < 3000)
  {
    motor[motorD] = 25;
    motor[motorE] = -25;
    currentDistance = (SensorValue[UltraSonic]);
    if (currentDistance < minimumDistance)
    {
      minimumDistance = currentDistance;
    }
  }
  stopMotors();
  wait1Msec(1000);
  while (SensorValue[UltraSonic] > minimumDistance + 5)
  {
    motor[motorD] = -25;
    motor[motorE] = 25;
  }
  stopMotors();
  nearStop(15, 25);
}
```

Extras: Make the back up and then seek the next nearest object

Example 2: Proportional Line Following

Variables for Line Following

```
// Variables for Line Following
int Kp = 1;
int offset = 45;
int Tp = 10;
int error;
int Turn;
int DPower;
int EPower;
```

Notes:
1. Kp is the Constant of Proportion. Multiply the "error" to increase the "sensitivity" of the Line follower.
2. offset is the threshold value for dark. Where the robot will travel straight.
3. Tp is the Constant for Power. This is the baseline power for the Left and Right wheels.

Method for Line Following:

```
// Proportional Line Follow Method
void LineFollow()
{
  error = offset - SensorValue[Light];
  Turn = Kp * error;
  DPower = Tp + Turn;
  if (DPower < 0)
  {
    DPower = 0;
  }
  if (Dpower > 60)
  {
    DPower = 60;
  }
  EPower = Tp - Turn;
  if (EPower < 0)
  {
    EPower = 0;
  }
  if (EPower > 40)
  {
    EPower = 40;
  }
  motor[motorD] = DPower;
  motor[motorE] = EPower;
}
```

Main Task for Line Following:

```
task main()
{
  servo[Arm] = 0;
  wait1Msec(2000);
  ClearTimer(T1);
  while (time1[T1] < 8000)
  {
    LineFollow();
  }
}
```

Example 3: Wall Following.

Using a proportional type algorithm, have the robot track along the wall and maintain a distance of about 40 centimeters.

```
// Wall Following Method
void wallFollow()
{
  int Kp = 1;
  int offset = 40;
  int Tp = 20;
  int error;
  int Turn;
  int DPower;
  int EPower;
  error = SensorValue[UltraSonic] - offset;
  Turn = error * Kp;
  DPower = Tp - Turn;
  if (DPower < 10)
  {
    DPower = 10;
  }
  if (DPower > 40)
  {
    DPower = 40;
  }
  EPower = Tp + Turn;
  if (EPower < 10)
  {
    EPower = 10;
  }
  if (EPower > 40)
  {
    EPower = 40;
  }

  motor[motorD] = DPower;
  motor[motorE] = EPower;
}
```

Use a timer and a while loop similar to Line Following to have the Wall Follow work in the main method.

Extras: Make the robot stop and scan forward to check for walls and corners.