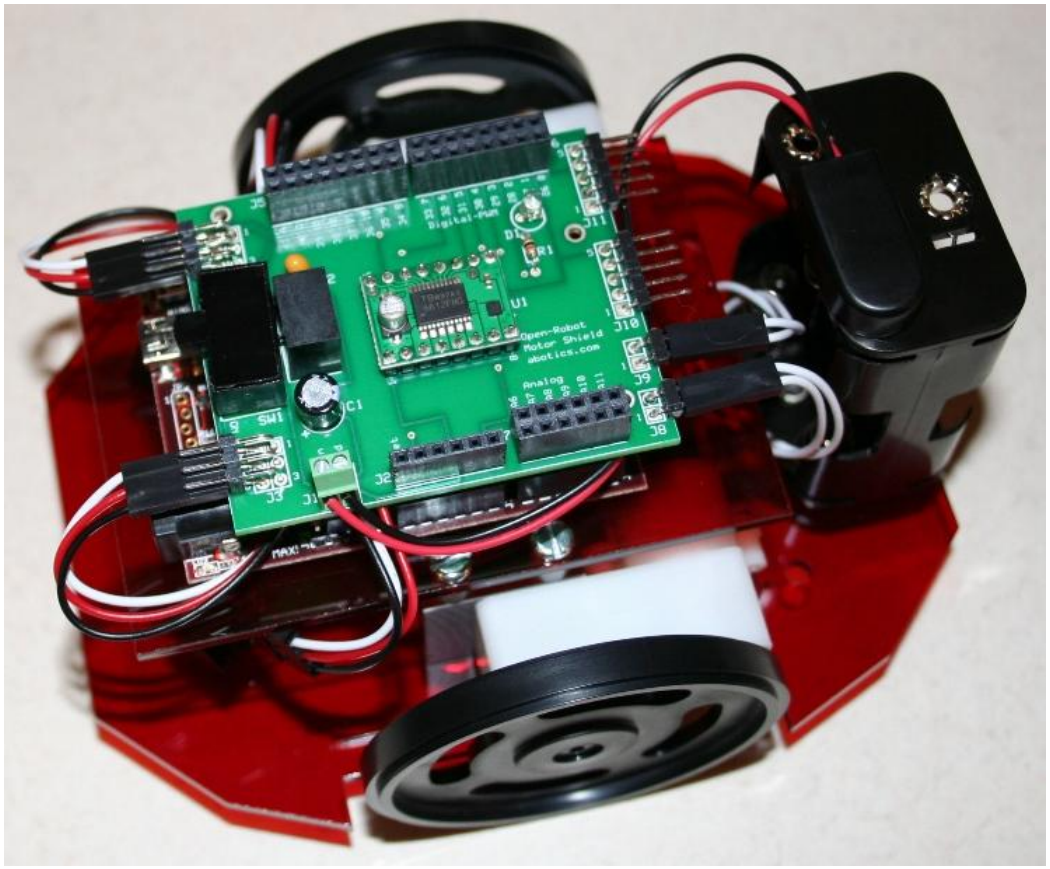

FIG 1. UNO32™ Based Open-Robot.

The PIC32™ based Open-Robot leverages the new open-source, chipKit UNO32™ Arduino™ compatible prototyping platform and opens up a whole new world of programming possibilities. A specially designed stackable circuit board provides motor control and the ability to read the state of (2) connected Sharp GP2Y0D810Z0F sensors. This board also accepts the new WW-12 Wheel Encoders from Nubotics (purchase separately) so that advanced motion control can be achieved. Future stackable, add-on boards will be developed and currently we have developed a wireless add-on module that will leverage the newly released RN-XV WiFi module from Roving Networks. Future potential add-on boards could encompass a camera board and/or RFID board.

We hope that end-users will develop their own special add-on boards since the hardware and software are open-source.

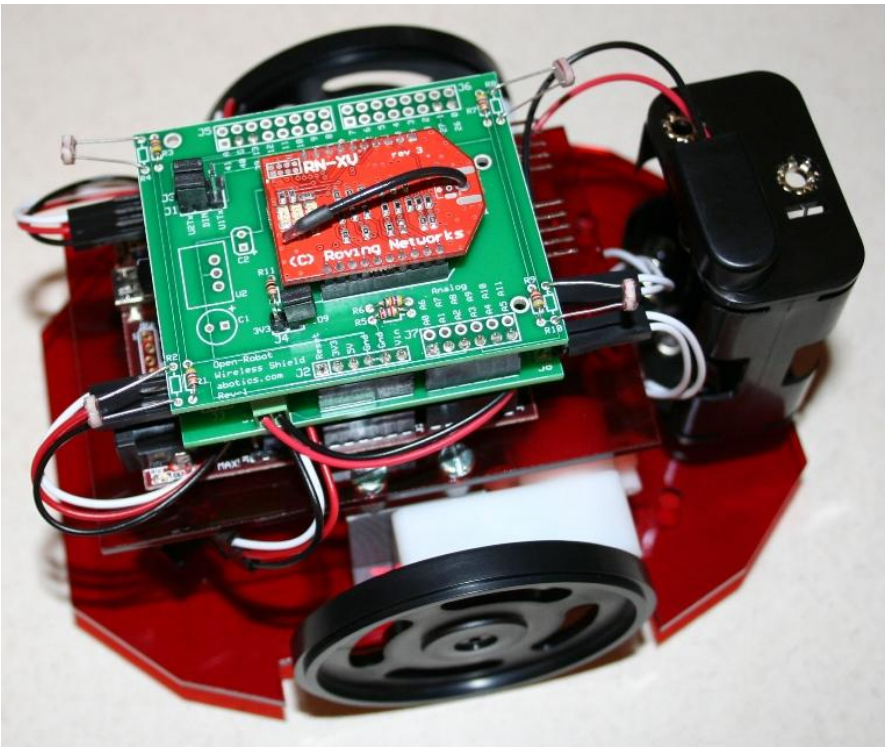

FIG 2. UNO32™ Based Open-Robot shown with wireless add-on board.

In Fig 2 above the wireless add-on board is shown installed on the Open-Robot base. The wireless board provides WiFi functionality to Open-Robot along with (4) light sensors that have been designed into voltage dividers and the resulting output voltage is fed into the PIC32's analog-to-digital (A/D) capable pins. In this way the voltage output is converted into a 10-bit number (0 – 1024). The intensity of ambient light is directly proportional to the coverted digital value.

INDEX

1 Getting Started with Your New 32-Bit Open-Robot

First, download the Arduino™ Multi-Platform IDE (MPIDE) and you will be able to create, compile and download new programs to your Open-Robot using a simple USB cable. No expensive in-circuit programmers to purchase unless you need to perform advanced level debugging. https://github.com/chipKIT32/chipKIT32-MAX/downloads

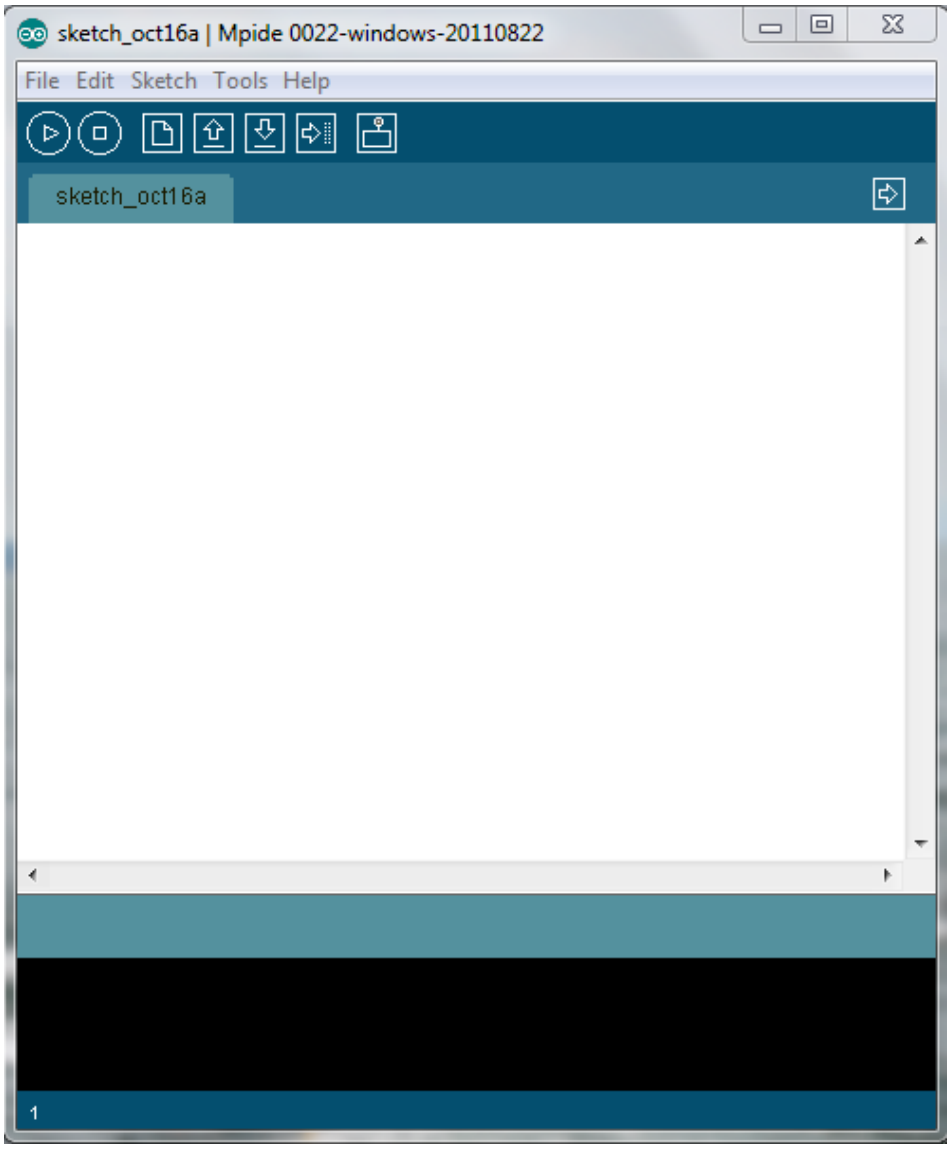

FIG 3. MPIDE Screen Shot.

After downloading and installing the MPIDE you should be able to run it and observe a window similar to the one shown above in Fig 3. Next download the latest library version from the Abe Howell's Robotics website. http://www.abotics.com/open_robot.htm

1.1 Working with Multi-Platform Integrated Development Environment (MPIDE)

From the MPIDE **File** menu select **Open** and browse for the example file that is contained within the library you just downloaded and after opening the MPIDE should display something similar to what is shown below in Fig 4.

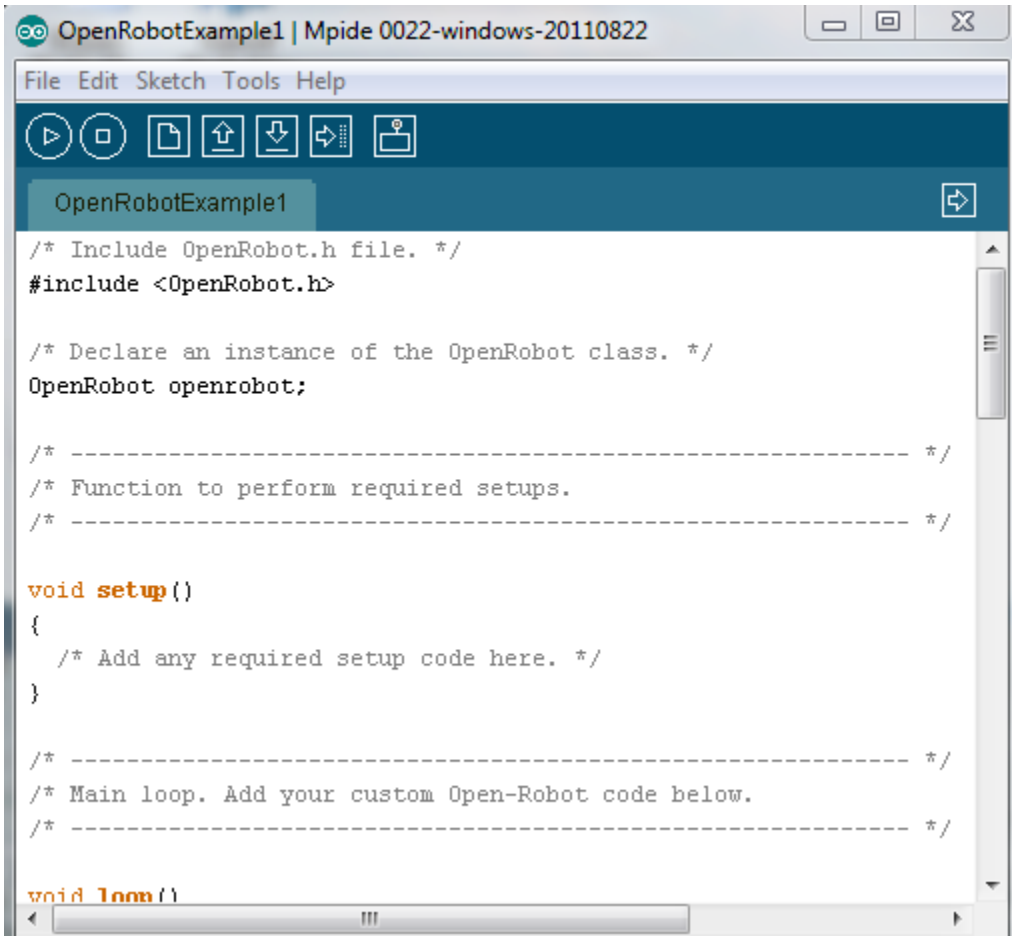

FIG 4. MPIDE displaying Open-Robot library example file, OpenRobotExample1.

Connect up the supplied USB programming cable to one of your computer's USB ports and connect the smaller end to the chipKit UNO32™ development board as shown below in Fig 5.

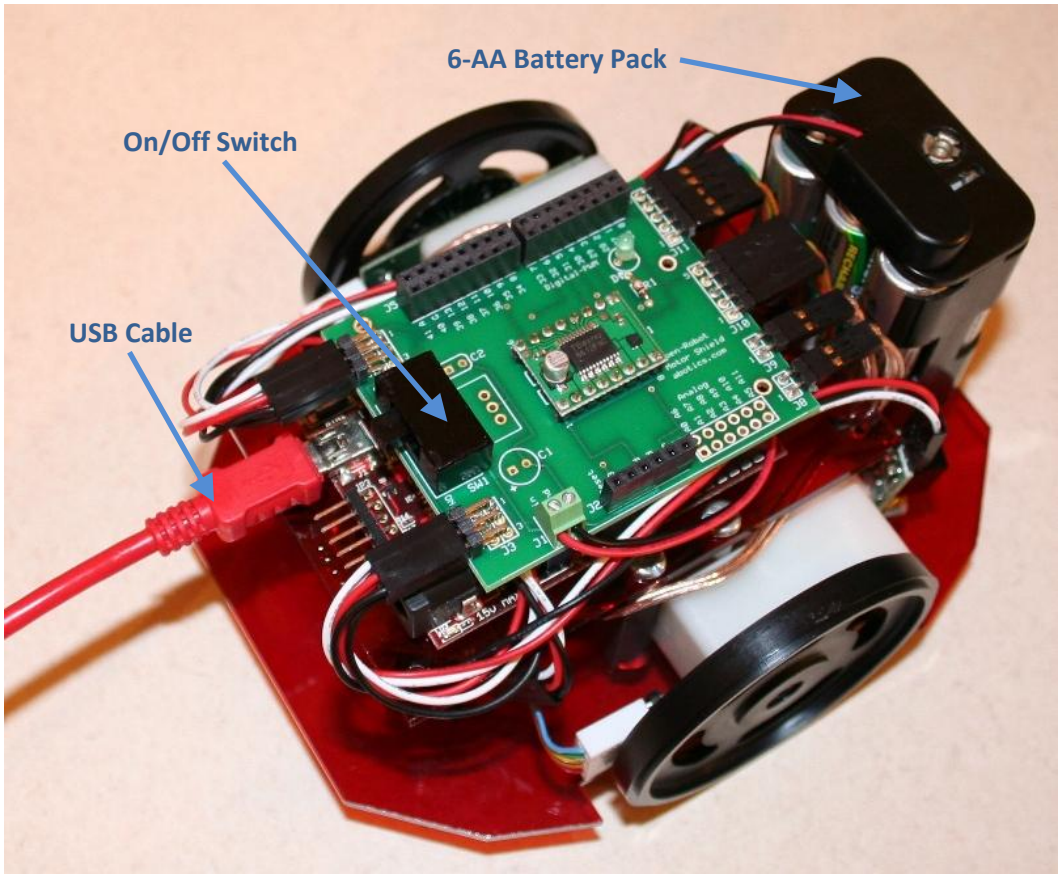

FIG 5. Connect USB programming cable to Open-Robot. Shown with optional WW02 Encoders & (2) additional IR sensors (purchase separately).

After connecting Open-Robot up to your computer the chipKit UNO32™ board should be recognized as a USB-to-Serial convertor and create a Virtual COM Port on your machine. Windows® users can find out what COM Port was created when the USB Serial Port was created. Refer to Fig 6 below. You can also select **Help → Getting Started** for more info.

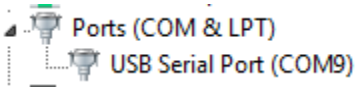

FIG 6. USB Serial Port was created on COM9, but will likely be different on your computer.

You now need to select the correct board from the **Tools → Board** menu. Be sure to select **chipKit UNO32**. You can refer to the built-in help by selecting **Help → Environment** and a webpage will open and display associated help contents. Go ahead and click on the compile button, which is the button that looks like a play symbol or **Sketch → Verify / Compile**.

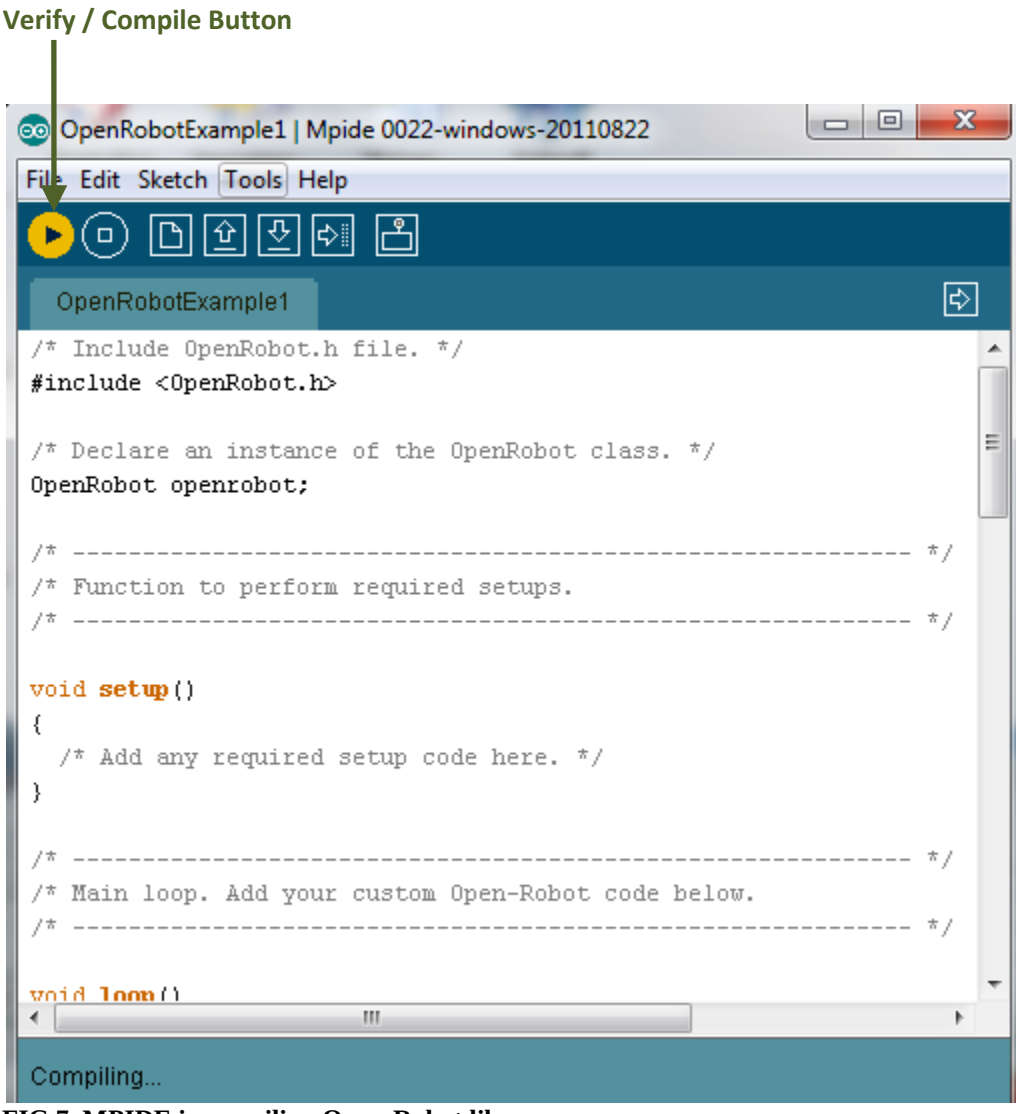

FIG 7. MPIDE is compiling Open-Robot library.

If the compile was successful, then the status window should display **Done Compiling**. You are now ready to upload the compiled program to Open-Robot. Simply click on the **Upload** button or **File → Upload to I/O Board**. Refer to Fig 8 & 9 below.

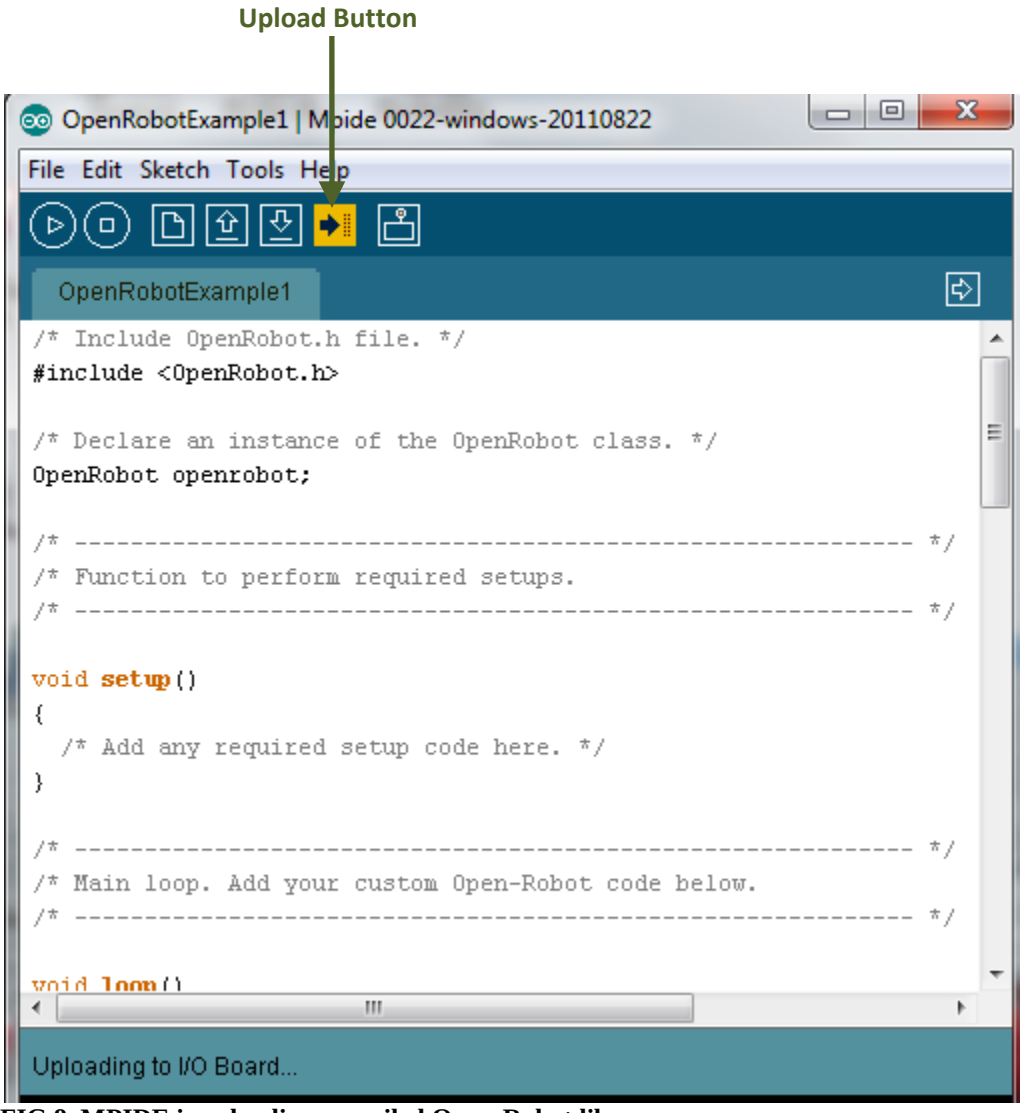

FIG 8. MPIDE is uploading compiled Open-Robot library.

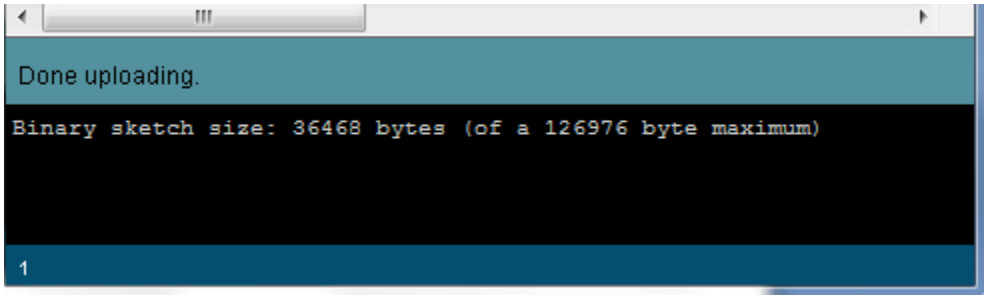

FIG 9. MPIDE upload is complete.

You have just successfully uploaded your first program to Open-Robot!

2 Basics of Open-Robot Library

Open-Robot's chipKit UNO32™ library is based around the 32-bit PIC32MX320F128H chip and was developed using the Arduino™ MPIDE. End-users can modify this library using the freely available MPIDE software and then upload the compiled results to Open-Robot via a USB cable connection. Most end-users will want to start by modifying the contents of the main loop section of code shown below in Fig 10. Every program must have a main loop even if there is no code contained within the function. The first thing you will notice is the declaration of several variables. Four variables are defined as Boolean and these will be used to store the current state of the attached front left and right and rear left and right GP2Y0D810Z0F sensor. Two additional short sized variables will be used to denote the left and right motor speeds. Next you will find a while() loop that continuously loops and executes the code contained within. Of course any active interrupt will pull the CPU away from the main loop in order to service the corresponding interrupt service routine (ISR).

Looking back at Fig 8 you will notice that the first line is a comment, but the second line is an **#include** statement that ensures we have the **OpenRobot.h** header file included so that we have

access to the corresponding library functions. On the fifth line you will see a declaration of the OpenRobot class, "**OpenRobot openrobot;**". Once declared we have access to all the public functions contained within. We will discuss these functions in the upcoming sections. In order to access these functions you need to preface each function with the instance, **openrobot**, and then follow with a dot, **.**, and then the corresponding function. The first function call is made on line#31 and calls the **IsUart2DataAvailable()** function to check for received serial characters. The details of each function can be found within the **OpenRobot.h** header file.

```
public:
   OpenRobot();
   unsigned short GetAnalogSensorReading(unsigned char sensorNumber);
   bool GetDigitalSensorReading(unsigned char sensorNumber);
   bool IsRobotHalted();
   bool IsUart2DataAvailable();
   bool IsMoveComplete();
   bool IsPerformOffsetTuningActive();
   void MotorOffsetTuning();
   signed short AdjustMotorSpeed(signed short motorSpeed, bool isLeftMotor);
   void SetMotorSpeed(short leftMotorSpeed, short rightMotorSpeed);
   void SetPosition(signed short posR, signed short posL);
   void SetWheelVelocity(signed short speed_L, signed short speed_R);
   void ParseUART2Data();
  private:
   void InitializeHardware();
   void InitializeMotionVariables();
   void ParseMotionCommand();
```

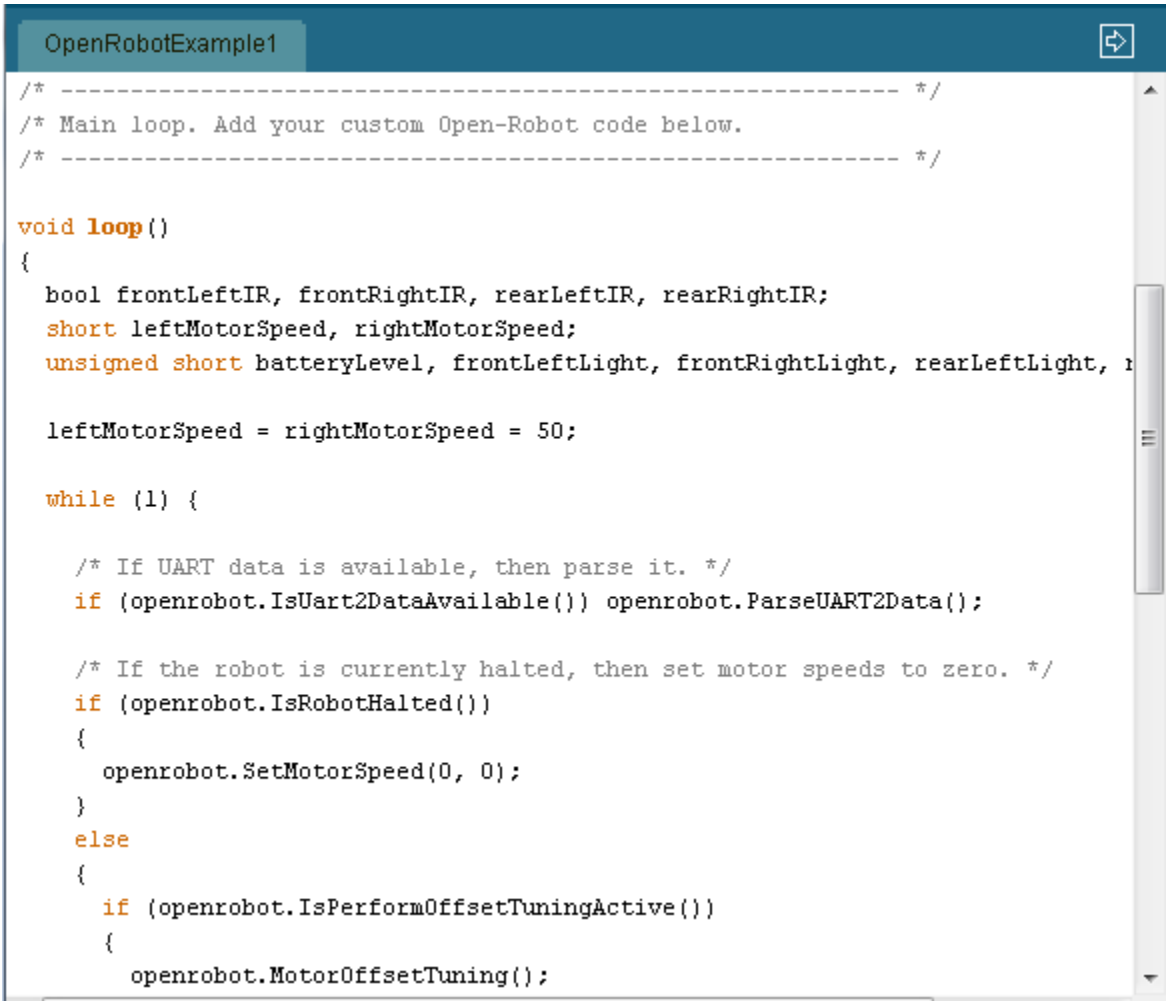

FIG 10. MPIDE screenshot of Open-Robot library example main loop.

Within the while loop you will see a section of code that leverages an If statement in order to determine whether the **IsRobotHalted()** returned Boolean state variable has been set by any part of the library and if so, then the robot's motors will be halted. When we say set with respect to a Boolean variable we mean that the variable has been set equal to 1 or true. When we say clear we mean the variable has been set equal to 0 or false.

2.1 Void SetMotorSpeed(short leftMotorSpeed, short rightMotorSpeed)

The **void SetMotorSpeed(short leftMotorSpeed, short rightMotorSpeed)** function provides the end-user with a function that sets the left and right motor speeds. The specific details of this function will be discussed later. For now the end-user can leverage this function and move the left and right wheels accordingly.

2.2 Void MotorOffsetTuning()

Within the Else statement section of code you will see a second If statement that is activated when the Boolean state variable performOffsetTuning is set. The **void MotorOffsetTuning()** function was created to find the motor offsets for both the left and right motors. This function requires properly installed WW-12 wheel encoders on both the left and right motors. Properly determining the motor offsets allows Open-Robot to drive as straight as possible and also helps the velocity and position control routines to run better.

In Fig 11 you can see that immediately after the **performOffsetTuning If** statement you will find another section of code that first acquires GP2Y0D810Z0F and light sensor readings for all attached sensors and then based upon the states prescribes appropriate left and right motor speeds.

```
/* Acquire up-to-date IR digital sensor readings. */
frontLeftIR = openrobot.GetDigitalSensorReading(0);
frontRightIR = openrobot.GetDigitalSensorReading(1);
rearLeftIR = openrobot.GetDigitalSensorReading(2);
rearRightIR = openrobot.GetDigitalSensorReading(3);

/* Acquire up-to-date analog battery & light sensor readings. */
batteryLevel = openrobot.GetAnalogSensorReading(0);
frontLeftLight = openrobot.GetAnalogSensorReading(1);
frontRightLight = openrobot.GetAnalogSensorReading(2);
rearLeftLight = openrobot.GetAnalogSensorReading(3);
rearRightLight = openrobot.GetAnalogSensorReading(4);

if (!frontLeftIR && !frontRightIR){
  openrobot.SetMotorSpeed(leftMotorSpeed, rightMotorSpeed);
}
else if (!frontLeftIR && frontRightIR){
  openrobot.SetMotorSpeed(-leftMotorSpeed, rightMotorSpeed);
}
else if (frontLeftIR && !frontRightIR){
  openrobot.SetMotorSpeed(leftMotorSpeed, -rightMotorSpeed);
}
else if (frontLeftIR && frontRightIR){
  openrobot.SetMotorSpeed(-leftMotorSpeed, -rightMotorSpeed);
}
```

FIG 11. MPIDE screenshot of Open-Robot library.

2.3 Bool GetDigitalSensorReading(unsigned char sensorNumber)

Recall previously that we defined four Boolean variables for holding the current state of the attached GP2Y0D810Z0F sensors. We can make use of the built-in, available digital pin read function, **bool GetDigitalSensorReading(unsigned char sensorNumber)** and obtain the state of each connected GP2Y0D810Z0F sensor. A standard Open-Robot is built with (2) GP2Y0D810Z0F sensors, but a total of (4) are supported. A returned 1 means blocked and a 0 means that the sensor is clear or unblocked.

2.4 Bool GetAnalogSensorReading(unsigned char sensorNumber)

```
/* Acquire up-to-date analog battery & light sensor readings. */
batteryLevel = openrobot.GetAnalogSensorReading(0);
frontLeftLight = openrobot.GetAnalogSensorReading(1);
frontRightLight = openrobot.GetAnalogSensorReading(2);
rearLeftLight = openrobot.GetAnalogSensorReading(3);
rearRightLight = openrobot.GetAnalogSensorReading(4);
```

FIG 12. MPIDE screenshot of GetAnalogSensorReading.

If we look at Fig 12 we will see a section of code that acquires the analog sensor readings for any connected light sensors or some other analog voltage input. After reading all the digital and analog sensors, the code then analyzes the sensor states and prescribes appropriate motor output values as shown below in Fig 13.

```
if (!frontLeftIR && !frontRightIR){
  openrobot.SetMotorSpeed(leftMotorSpeed, rightMotorSpeed);
}
else if (!frontLeftIR && frontRightIR){
  openrobot.SetMotorSpeed(-leftMotorSpeed, rightMotorSpeed);
}
else if (frontLeftIR && !frontRightIR){
  openrobot.SetMotorSpeed(leftMotorSpeed, -rightMotorSpeed);
}
else if (frontLeftIR && frontRightIR){
  openrobot.SetMotorSpeed(-leftMotorSpeed, -rightMotorSpeed);
}
```

FIG 13. MPIDE screenshot of code that analyzes sensor readings and generates motor speeds.

Above in Fig 13 we can see that there are a total of four conditionals contained within the If and Else If statements. The first statement checks for the front left and right sensors to be clear and if this is true then it will set the left and right motor speed so that the robot drives forward. The second statement checks for the front right sensor to be blocked, the front left to be clear and then it sets the motors to turn the robot away from the detected obstacle. The third statement checks for the front left sensor to be blocked, the front right to be clear and then it sets the motors to turn the robot away from the detected obstacle. The fourth statement checks for both

the front left and right sensors to be blocked and then sets the motors to reverse the robot away from the detected obstacle(s).

It is now recommended that you now change the signs of the left and right motor speeds, **leftMotorSpeed** & **rightMotorSpeed**, recompile, upload and then observe how the robot's behavior changes. Create an entirely new behavior that keeps the robot still until one or both of the front sensors are blocked. When one of the sensors is blocked the robot should backup while turning away from the detected obstacle. If both sensors are blocked then the robot should back straight up.

3 Details of Wireless Circuit Board

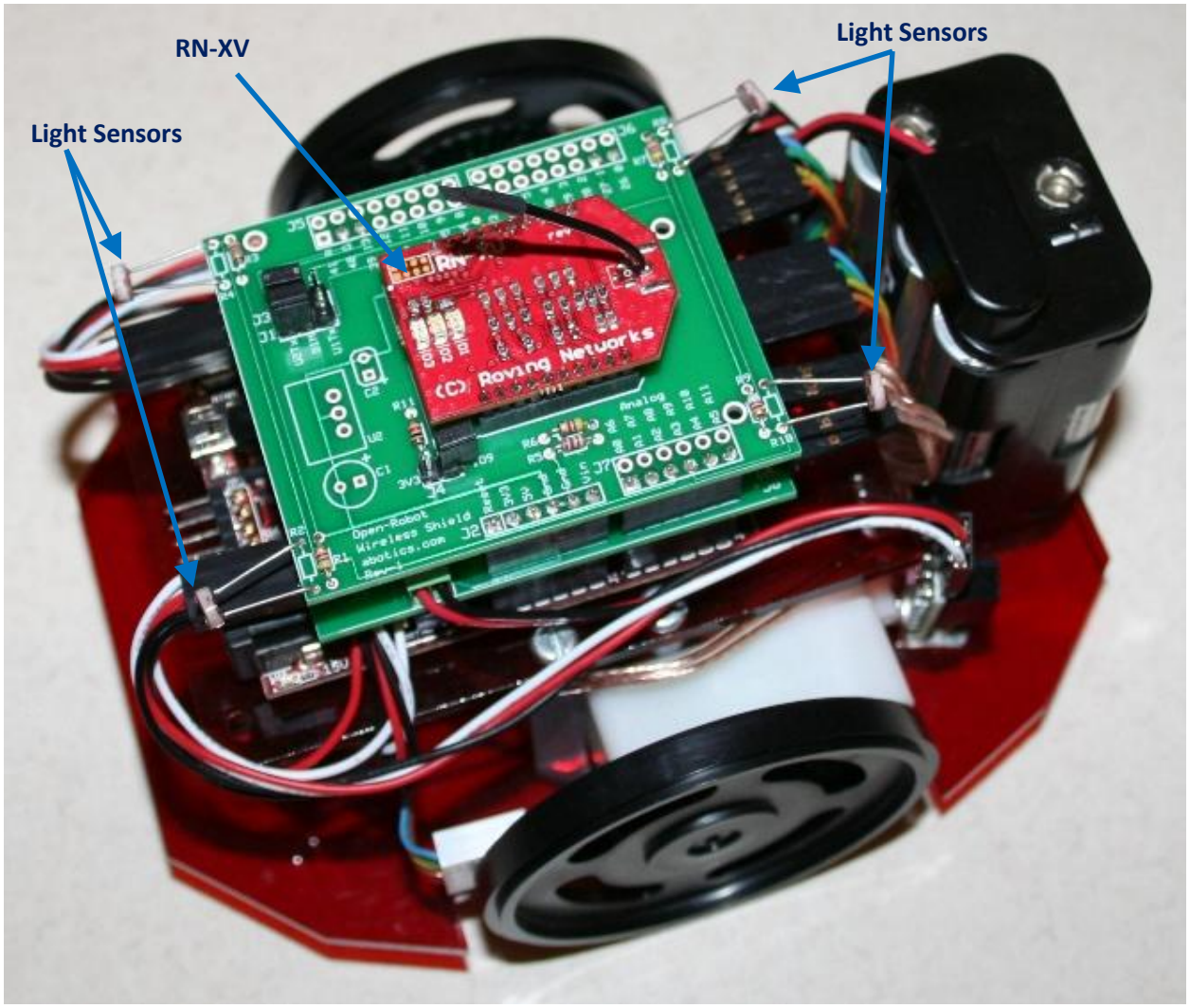

FIG 14. Open-Robot Wireless board shown with RN-XV WiFi module and (4) light sensors.

Adding the wireless circuit board provides Open-Robot with the ability to communicate with another WiFi enabled device such as a laptop, PC or another WiFi enabled robot. The wireless board has a total of (4) cadmium sulfide photo-resistor cells (light sensors) that are wired into a voltage divider with appropriately sized resistors (4.7 kOhm).

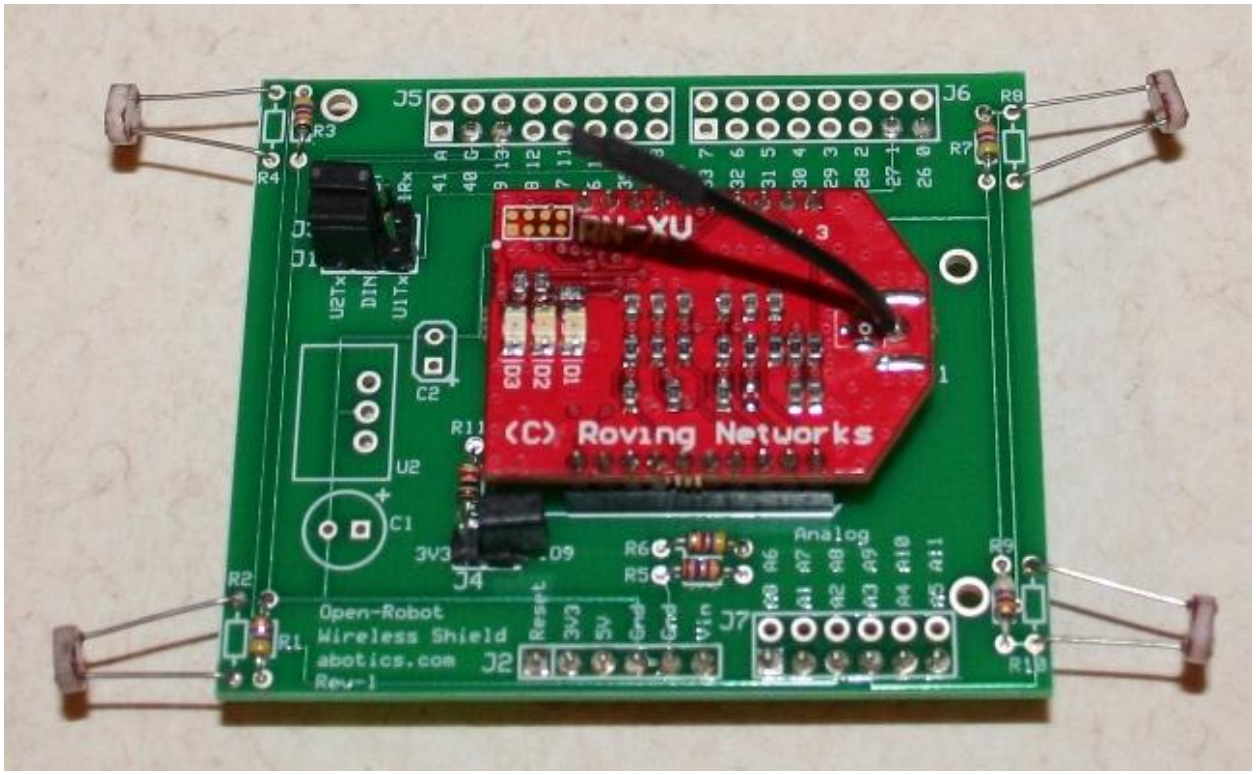

FIG 15. Open-Robot wireless PCB up close.

In Fig 15 above it can be seen that there are a total of (3) sets of jumpers: J1, J3 and J4. J1 and J3 specify which UART pins connect to the RX-NV WiFi module's DIN and DOUT pins. In general these should remain connected to U2Rx-DOUT and U2Tx-DIN since UART1 is connected to the FTDI USB chip and facilitate the uploading of new programs from MPIDE.

3.1 Enter Ad-Hoc Mode on RN-XV WiFi Module

J4 can be used to reboot the RN-XV in Ad-Hoc mode. Prior to switching Open-Robot on place the jumper on J4 and connect +3.3 volts to PIO9 on the RN-XV module. Switch Open-Robot on and the RN-XV will boot-up in Ad-Hoc mode. If you browse for available wireless networks you should find the WiFly-GSX-85 Ad-Hoc network that corresponds to the RN-XV and then connect to it. Once connected you can connect using TeraTerm, which must be downloaded from Roving Network's website: http://rovingnetworks.com/products/RN_XV

You should also download the corresponding user manual, "WiFly-RN-UM.pdf", in case you are interested in advanced configurations of the RN-XV module or want to configure it for Infrastructure mode so that it can access your specific wireless access point.

3.1.1 Connect to RN-XV Using Tera Term

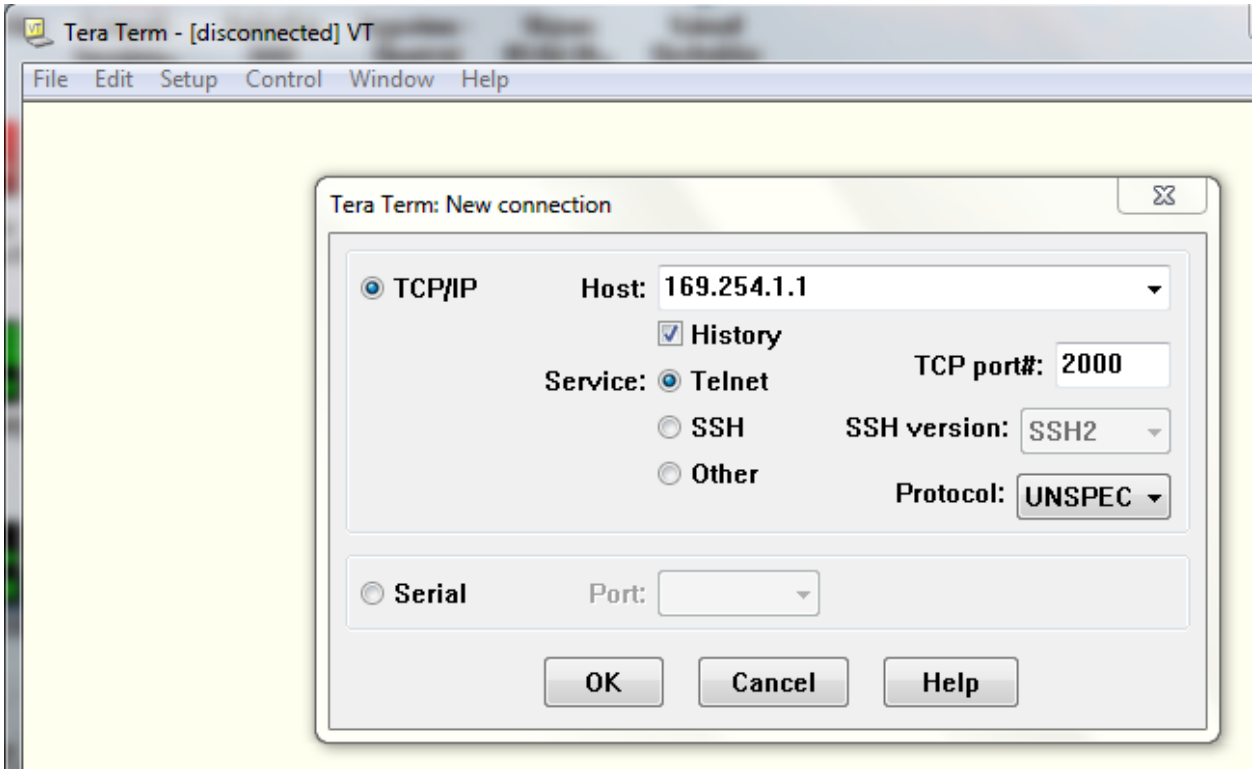

FIG 16. TeraTerm Ad-Hoc connection.

After launching TeraTerm you need to select TCP/IP, Telnet, TCP port# 2000 and enter Host Address: 169.254.1.1 as shown above in Fig 16.

You can verify that your computer has properly connected to the RN-XV by using the ***ping*** command from a command prompt as shown in Fig 17 below. You can get to the command prompt by typing ***cmd*** from the Start Menu.

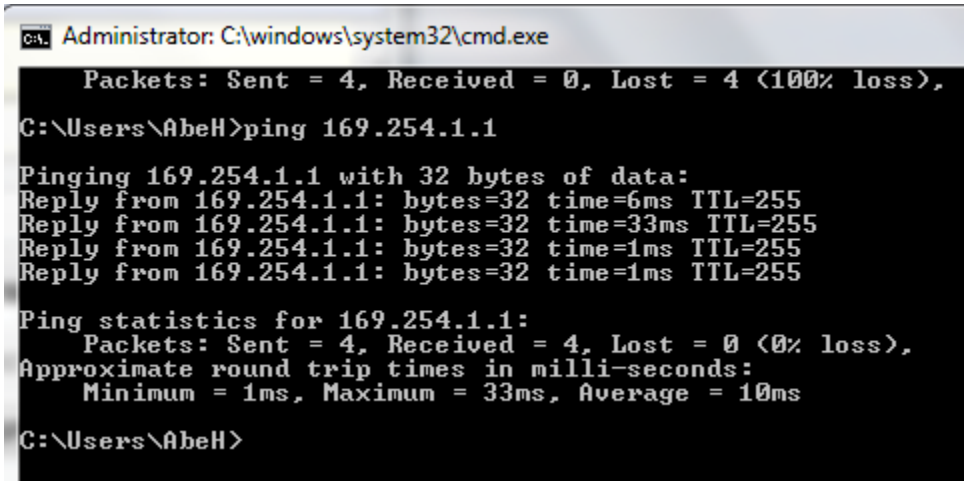

FIG 17. Command prompt session using ping 169.254.1.1.

Once the Ad-Hoc connection has been verified select OK in TeraTerm and then the *HELLO* message from the RN-XV should be printed as shown in Fig 18 below.

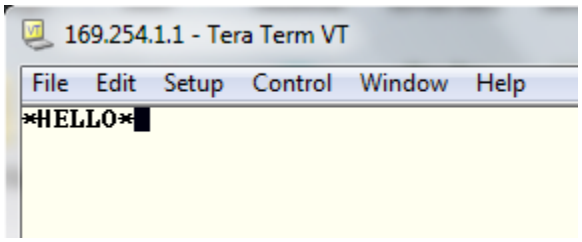

FIG 18. Connection response from RN-XV module.

3.1.2 Enter Command Mode

Enter command mode on the RN-XV module by typing **$$$.** You will obviously need to hold the Shift key while pressing the number 4 ($) key three times. The RN-XV will respond with **CMD** as shown below in Fig 19.

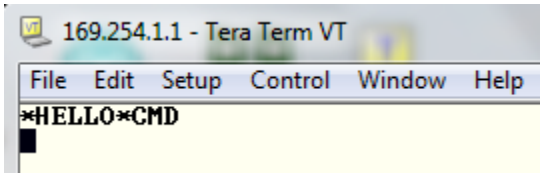

FIG 19. Enter command mode on RN-XV module.

Once in command mode you can read and write numerous configuration parameters as shown in the RN-XV user manual. One useful command is **get everything**. After typing get everything and hitting the Enter key you will notice that the RN-XV responds by printing out all of the relevant configuration parameters as shown in Fig 20.

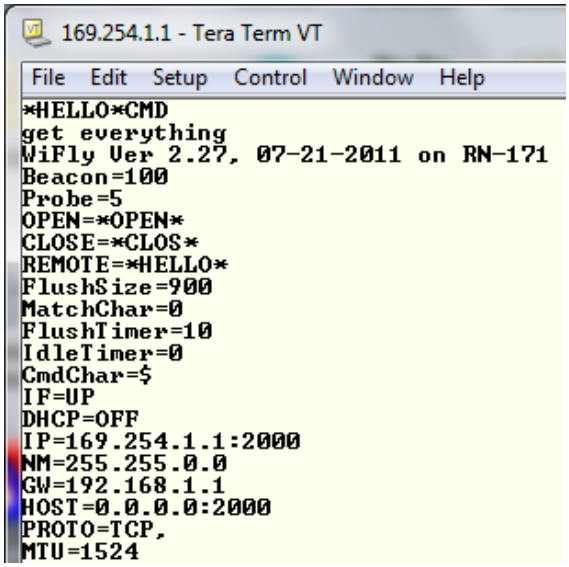

FIG 20. Get Everything command printout from RN-XV.

Two additional commands that might be useful are **get uart baud rate** and **set uart baudrate**. These two commands are fairly obvious and allow you to get and set the baudrate that is used by the RN-XV on the DIN and DOUT lines. Open-Robot leverages a 115,200 baudrate with 8-data bits, 1-stop bit and no parity. From Fig 21 below it can be seen that the RN-XV module is configured with a 115200 baudrate.

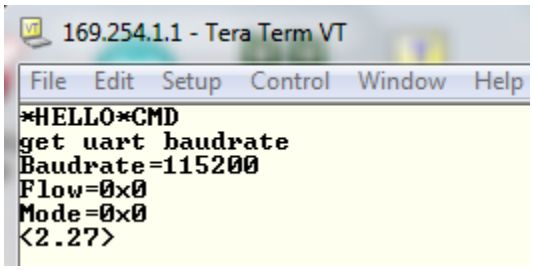

FIG 21. Get Uart Baudrate command printout from RN-XV.

After setting the baudrate or changing any other parameters you will need to save the configuration using the **save** command. You can exit command mode by simply typing the exit command as shown in Fig 23.

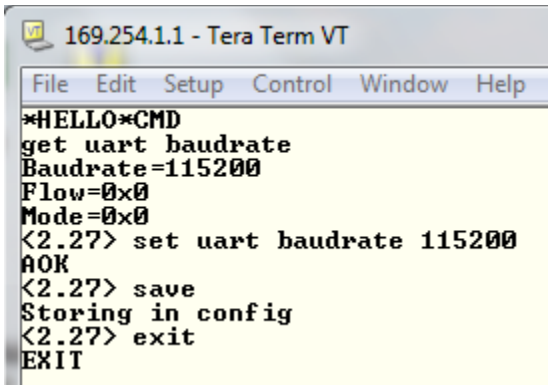

FIG 23. Save and exit command printout from RN-XV.

4 Wireless Control – Firmware Command Set

A simple ASCII readable command protocol has been built into Open-Robot's PIC32 firmware. This easy-to-use command set provides the end-user with a mechanism for communicating and controlling Open-Robot wirelessly. In order for this to be possible you must have purchased the add-on wireless PCB that is described in chapter 3.

4.1 Available Wireless Commands

All commands leverage ASCII human-readable characters, so that an end-user can easily send and receive data and be capable of interpreting them. Command characters are contained with the double-quotes, **"**. Do not send the double-quotes.

4.1.1 Get analog sensor readings – "A\r"

a. This command will return analog sensor readings such as battery level and light sensor readings. Response appears as shown next: **a139,200,198,150,138/r**

b. All readings are comma-delimited and in the following order: battery level, front left light, front right light, rear left light and rear right light.

c. For this command to function properly you must have the WiFi add-on board installed otherwise you will acquire meaningless readings.

4.1.2 Get digital sensor readings – "D\r"

d. This command will return digital sensor readings such as those from the Sharp GP2Y0D810Z0F IR sensors.

e. Response appears as shown next: **d1,0,0,1/r**

f. All readings are comma-delimited and in the following order: front left IR, front right IR, rear left IR and rear right IR sensor reading.

g. In general Open-Robot is shipped with only the two front IR sensors and you must purchase and install the two additional rear IR sensors otherwise the rear readings have no meaning.

4.1.3 Get encoder readings – "E\r"

h. This command will return the left and right wheel encoder readings.

i. Response appears as shown next: **e144,143/r**

j. All readings are comma-delimited and in the following order: left encoder count, right encoder count.

k. You must purchase and install the recommended WW02 Wheel Encoders for this command to provide meaningful readings.

4.1.4 Halt/stop robot – "H\r"

l. This command will immediately stop Open-Robot regardless of what command was sent prior. If in the middle of performing a closed-loop position or velocity command it will immediately stop the robot.

m. Command response is as follows: **h/r**

4.1.5 Motion Command – "Mxy,z\r"

n. This command is used to command Open-Robot to perform any one of three different motion types: Closed-Loop Position control, Closed-Loop Velocity control or Open-Loop Speed control.

o. Replace the **x** above with a **P** for Position control, a **V** for Velocity control and an **O** for Open-Loop Speed control.

p. The **y** above corresponds to the left motor and the **z** to the right motor.

q. If we want to have the left motor rotate 100 encoder ticks forward and the right motor rotate -100 encoder ticks using Position control, then we would send the following command: "**MP100,-100/r"**

r. Allowable range for Position control is [-500,500], for Velocity control it is [-150,150], and Speed control [-150,150].

4.1.6 Motor PWM Offset Tuning – "O\r"

s. This command will perform motor PWM offset tuning and eliminate PWM dead-bands for both the left and right motors. This command should only be used if you have purchased and installed the WW02 Wheel Encoders.

t. Command response is as follows: **o/r**

4.1.7 Get wheel velocities – "V\r"

u. This command will return the most current wheel velocity for the left and right wheel. You must have purchased and installed the WW02 Wheel Encoders for this function to provide meaningful readings.

v. Command response is as follows: **v25,-26/r**

w. All readings are comma-delimited and in the following order: left wheel velocity, right wheel velocity.