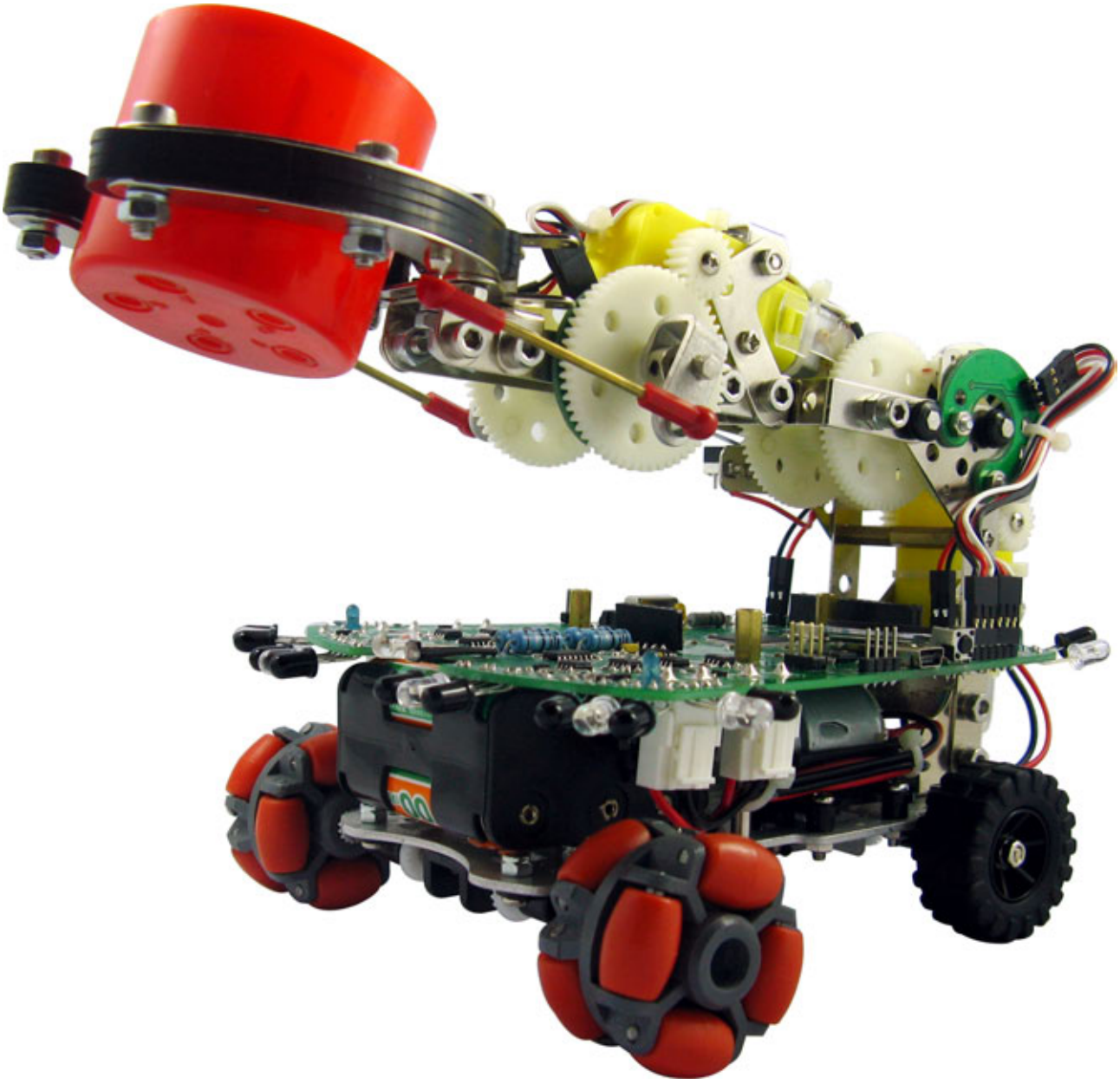
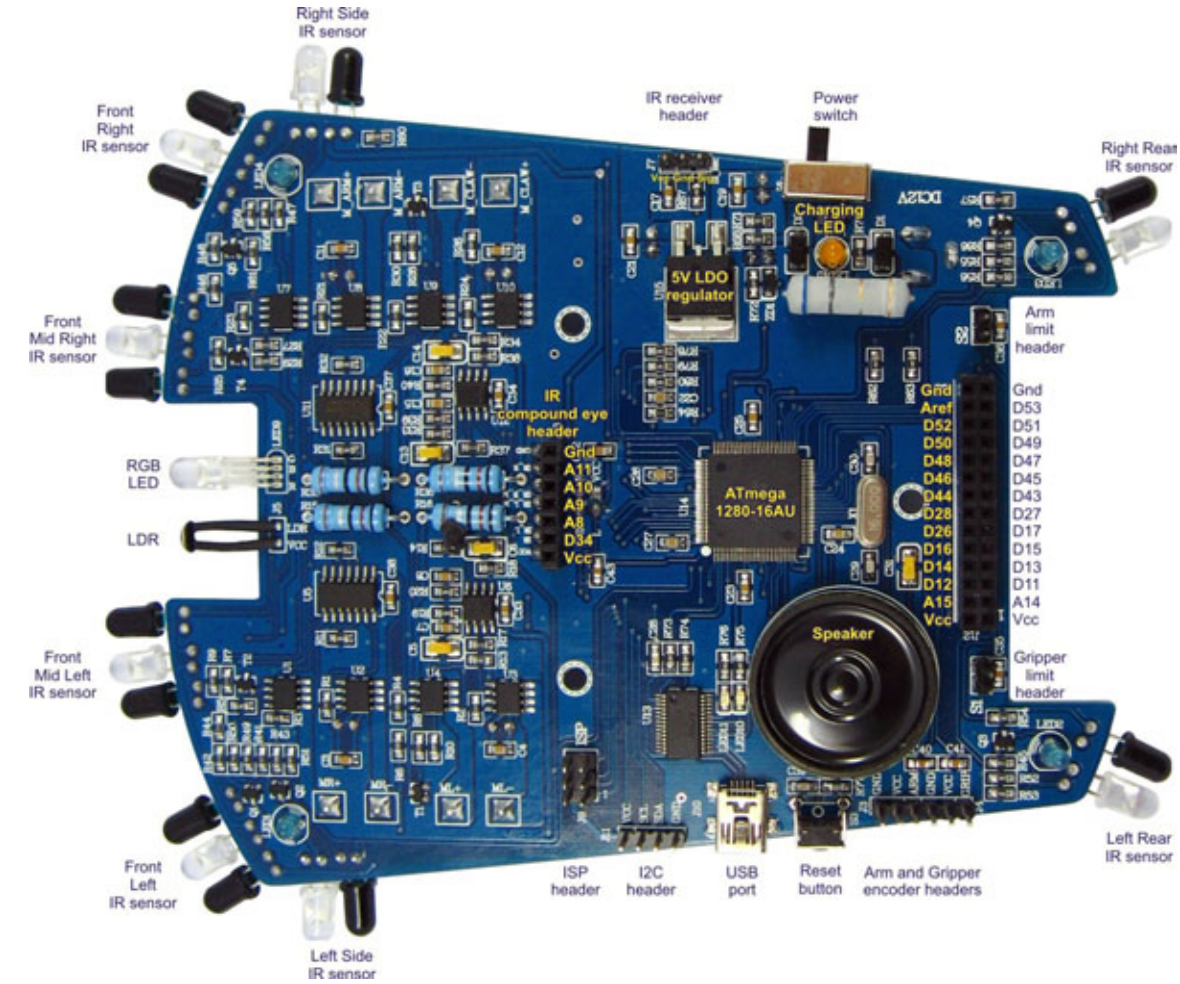
Mr.Tidy

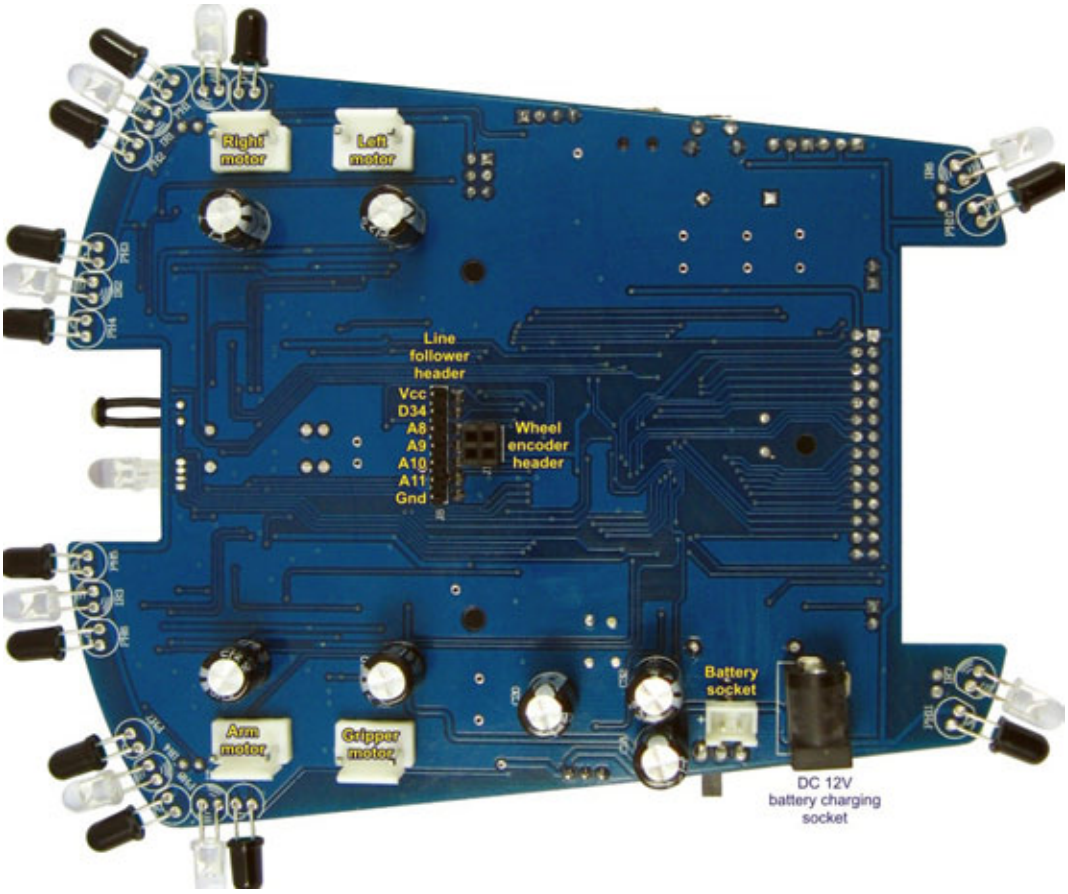


Mr. Tidy is a unique and versatile robotic platform ideal for students and hobbyist alike. The 2DOF arm and array of sensors make this robot capable of so much more than just line following and object avoidance. An open framework makes experimentation and expansion quick and easy.

Contents:

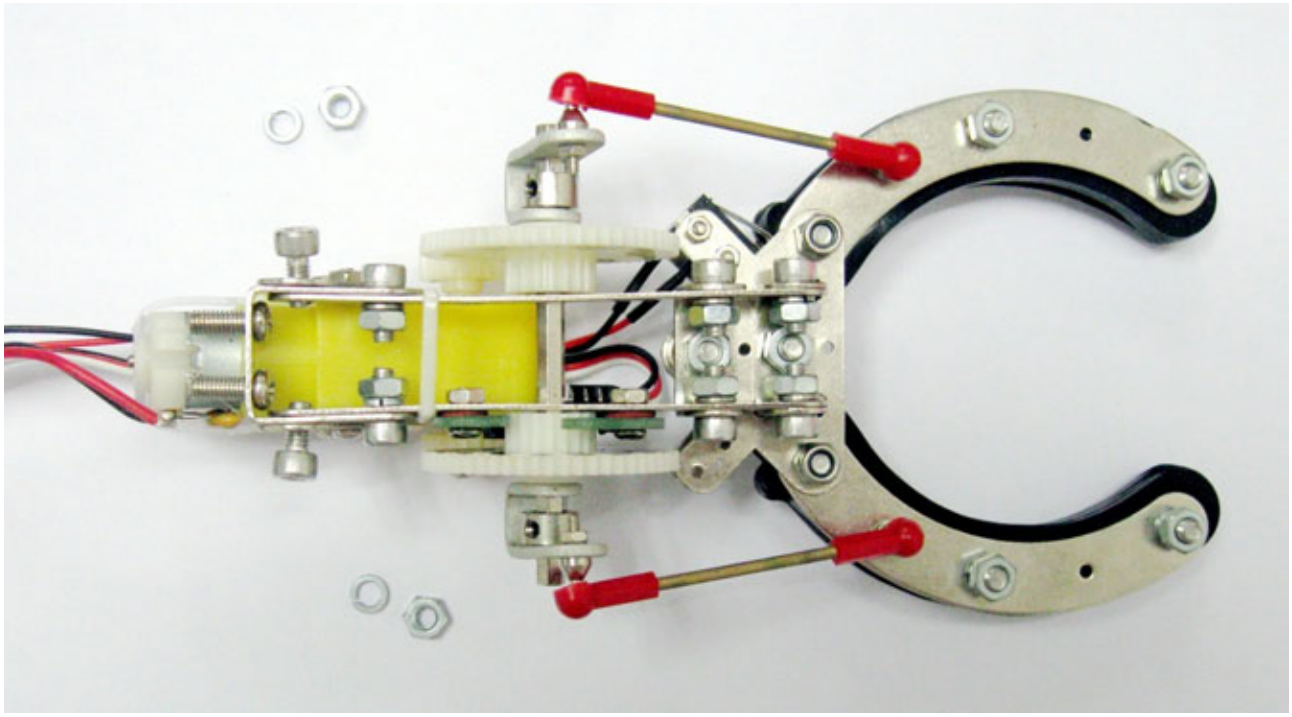
Main board Features



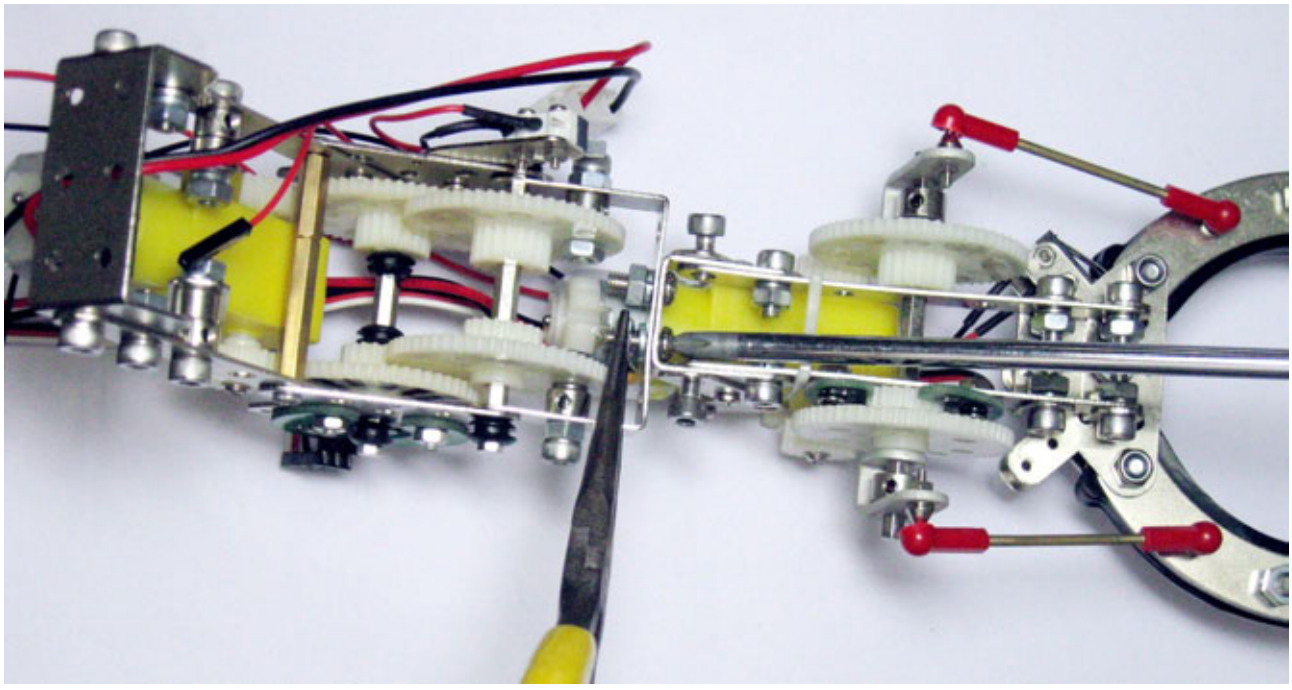


Assembly Instructions

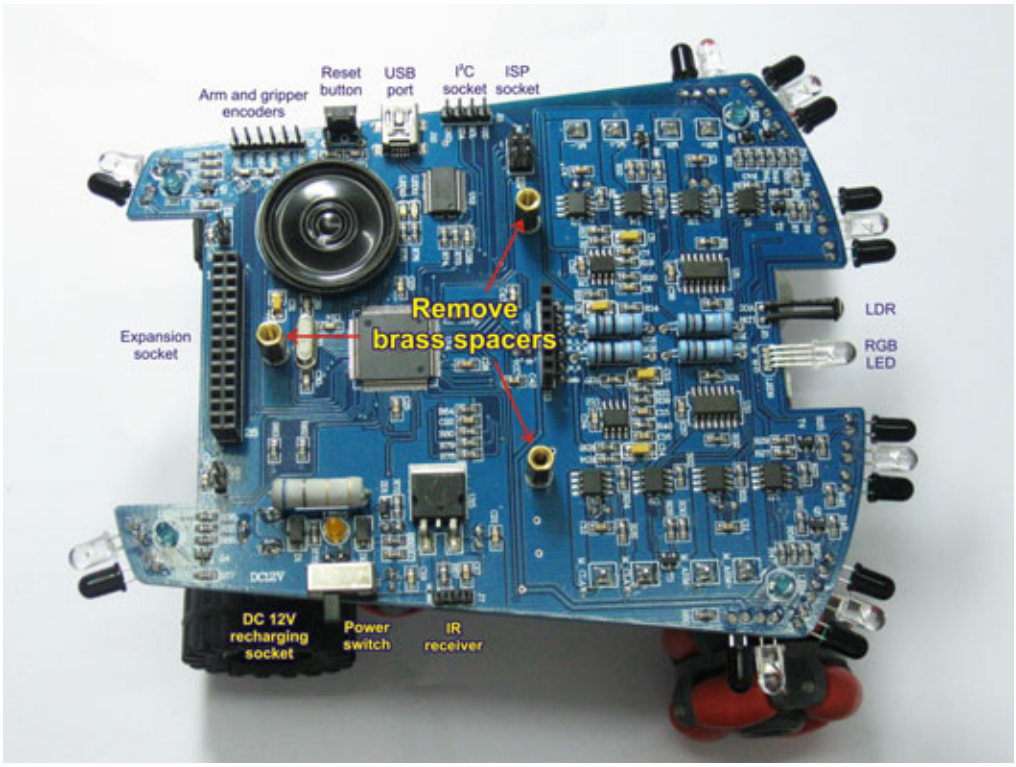
STEP 1: The robot arm comes in two parts that must be joined together before the arm is attached to the base. Start by removing the nuts and spring washers shown in the photo bellow. Do not completely remove the screws. This will allow you to use a screwdriver to join the two sections.



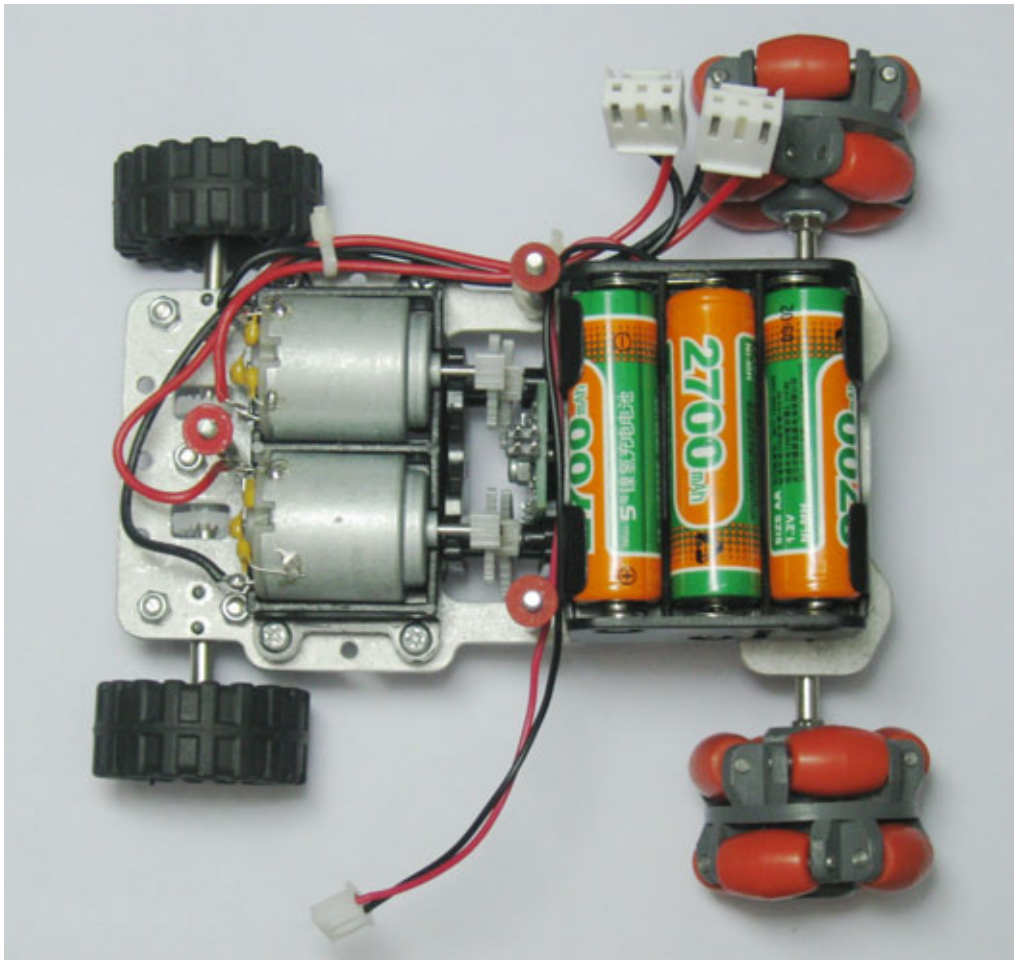
STEP 2: Join the two parts as shown below and then replace the washers and nuts removed in STEP 1.



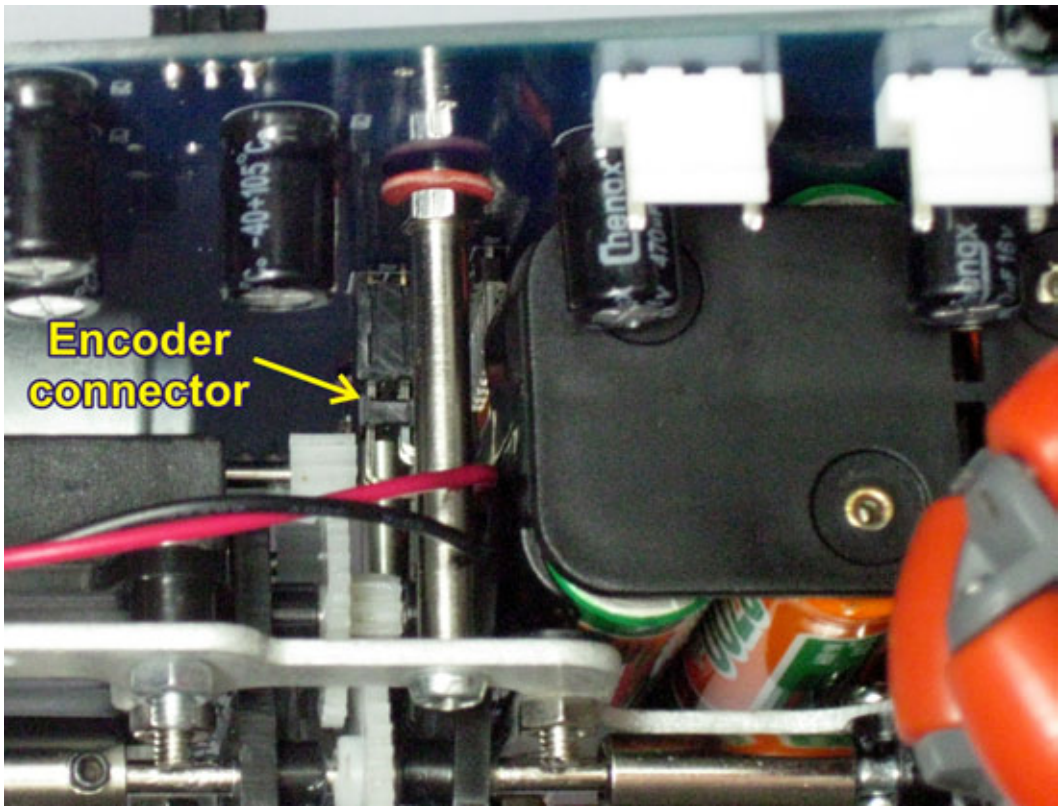
STEP 3: Undo the 3 brass spacers and remove the main board.



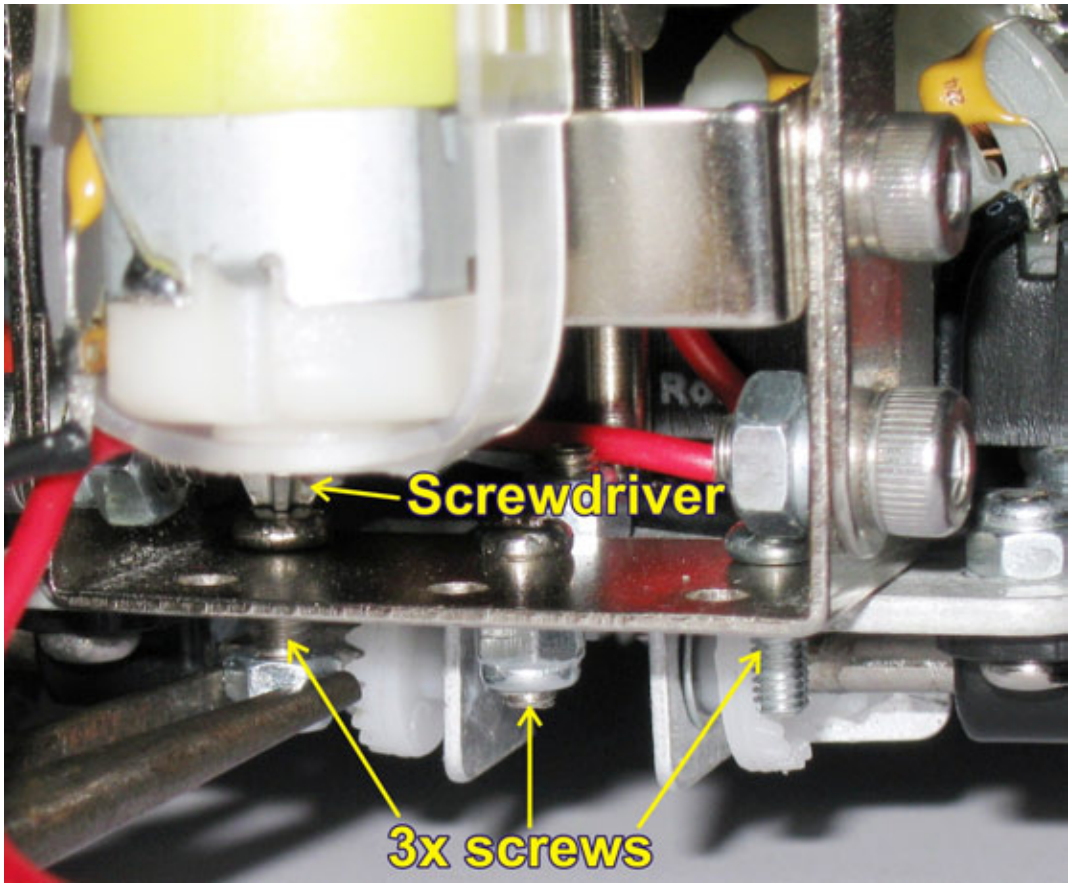
STEP 4: Insert 6x “AA” (UM3) NiMh batteries. Do not use alkaline batteries as they cannot supply enough power for the robot and cannot be recharged. Note how the battery cable is run.



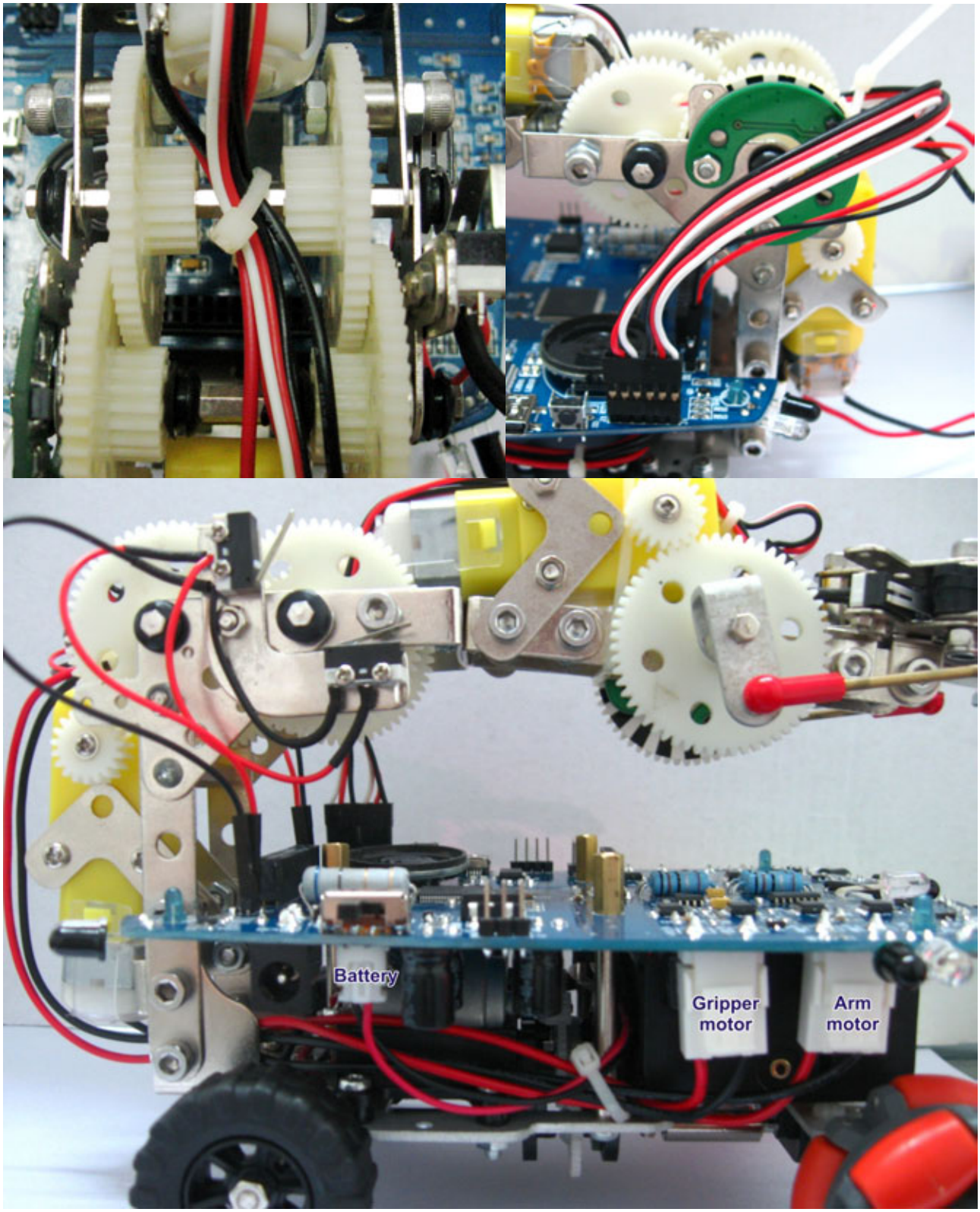
STEP 5: Re-install the main board being careful to ensure the encoder PCB for the wheels plugs in correctly. Make sure the battery cable is behind the mounting post so it cannot jam in the gears.



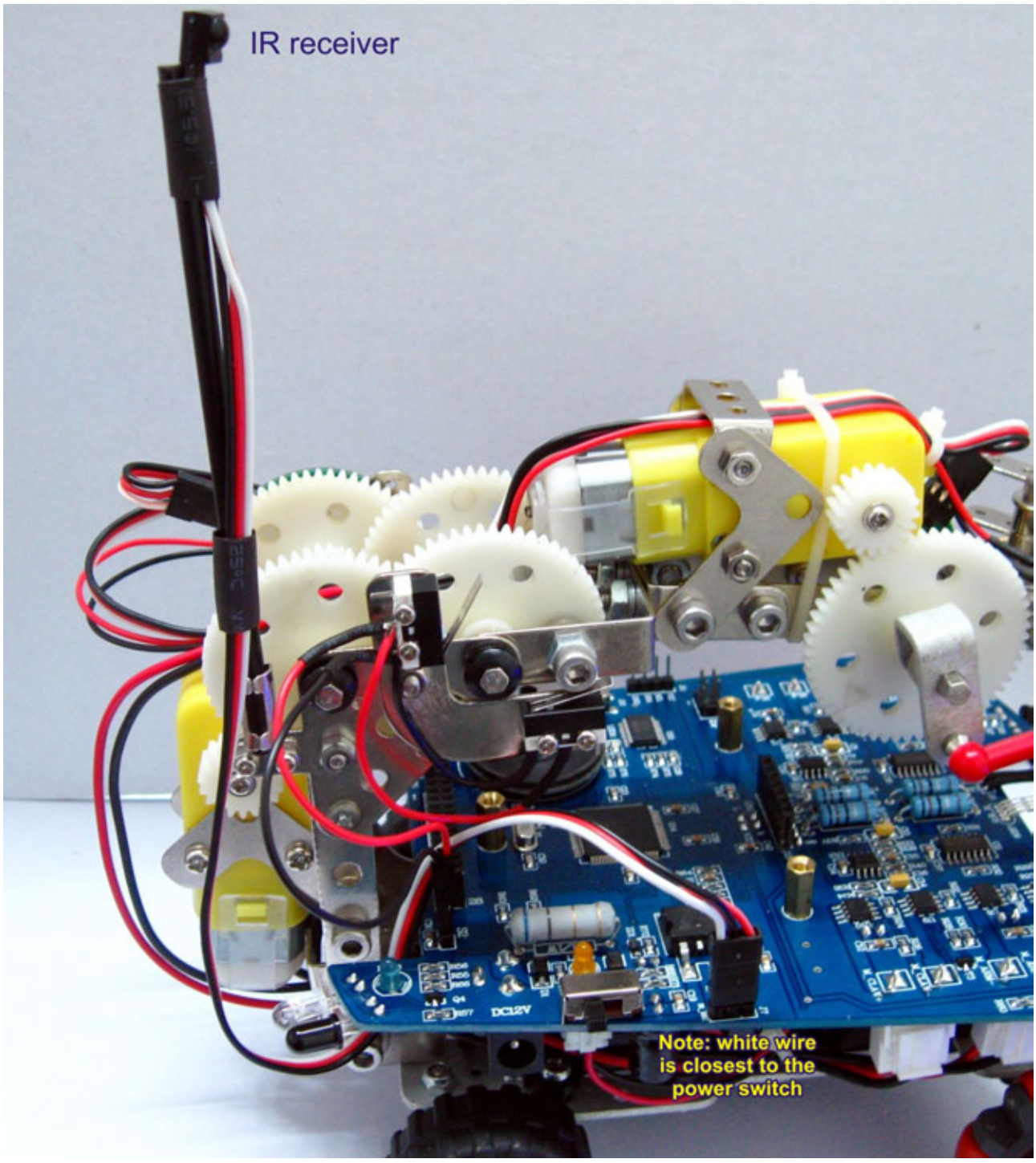
STEP 6: Mount the arm onto the base with 3x8mm screws as shown. The screwdriver goes between the gears from above. It may be easier to use pliers to hold the nut while starting the thread. A spanner can then be used to tighten the nuts.



STEP 7: Now that the arm is mounted we need to tie the wires to the frame and connect all the wires to the main board. Pay careful attention to the orientation of the plugs.



STEP 8: Attach the IR receiver to the robot. Pay careful attention to how the cable plugs into the processor board. Incorrect connection may damage the receiver.



STEP 9: Check your connections carefully before turning the robot on. When you turn it on it should start by playing a simple tune. The robot will then open its gripper, lower the arm until the lower limit switch is hit and then raise it's gripper. The robot will now search for objects to pick up.

STEP 10: optional sensor alignment rings can be fitted to help maintain sensor alignment when the robot bumps into something. These rings may reduce the sensitivity of the sensors slightly but can improve reliability.

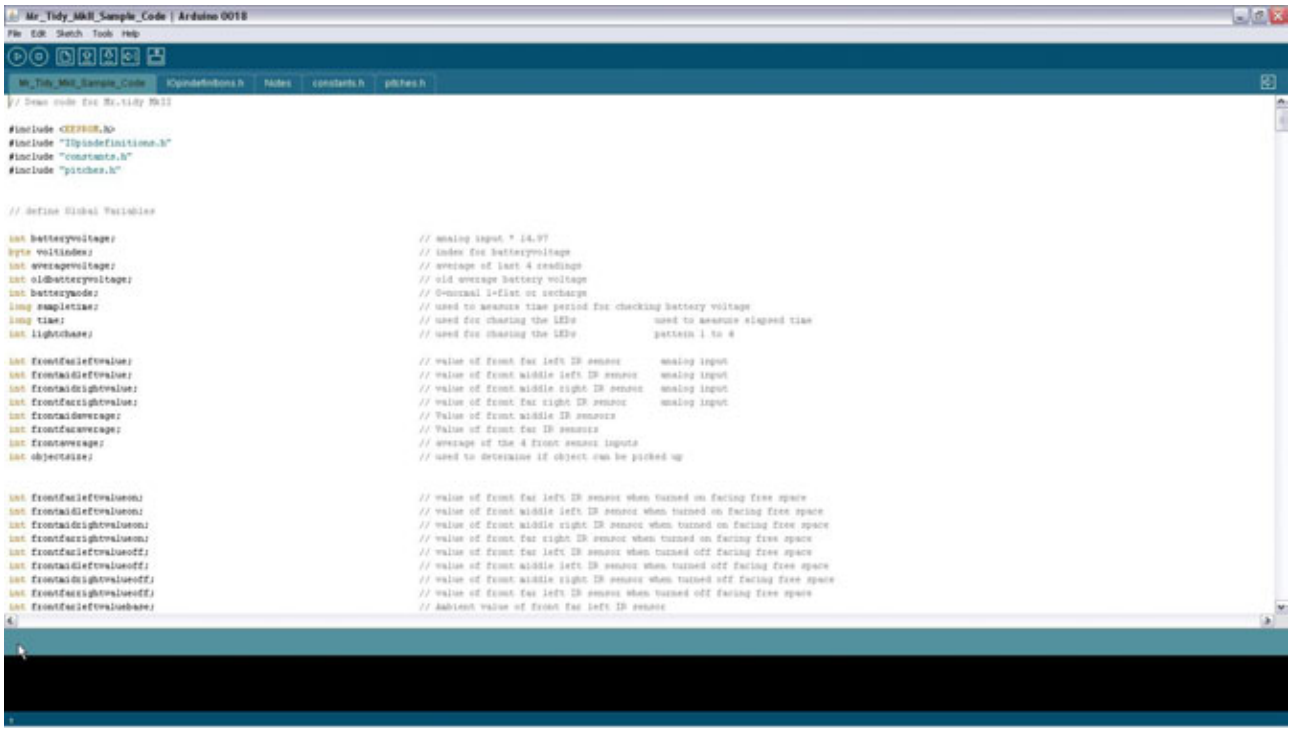
Installing the software

This robot can be programmed using a free **I**ntegrated **D**evelopment **E**nvironments (IDE) from the internet. Depending on your programming skills there are several choices. Many advanced programmers would choose an IDE such as AVR Studio and use the ISP socket to program this robot.

To make this robot as easy as possible to program we have chosen to make it compatible with the Arduino IDE and have included a USB interface. This manual assumes that the Arduino IDE (version 18 or later) is being used. Although the robot comes with the Arduino bootloader and sample software already installed you will need the Arduino IDE for loading the diagnostic software and writing your own code.

At the time this manual was written the Arduino IDE was available for Windows, Mac OS X and Linux 32bit. It can be downloaded from here: http://arduino.cc/en/Main/Software

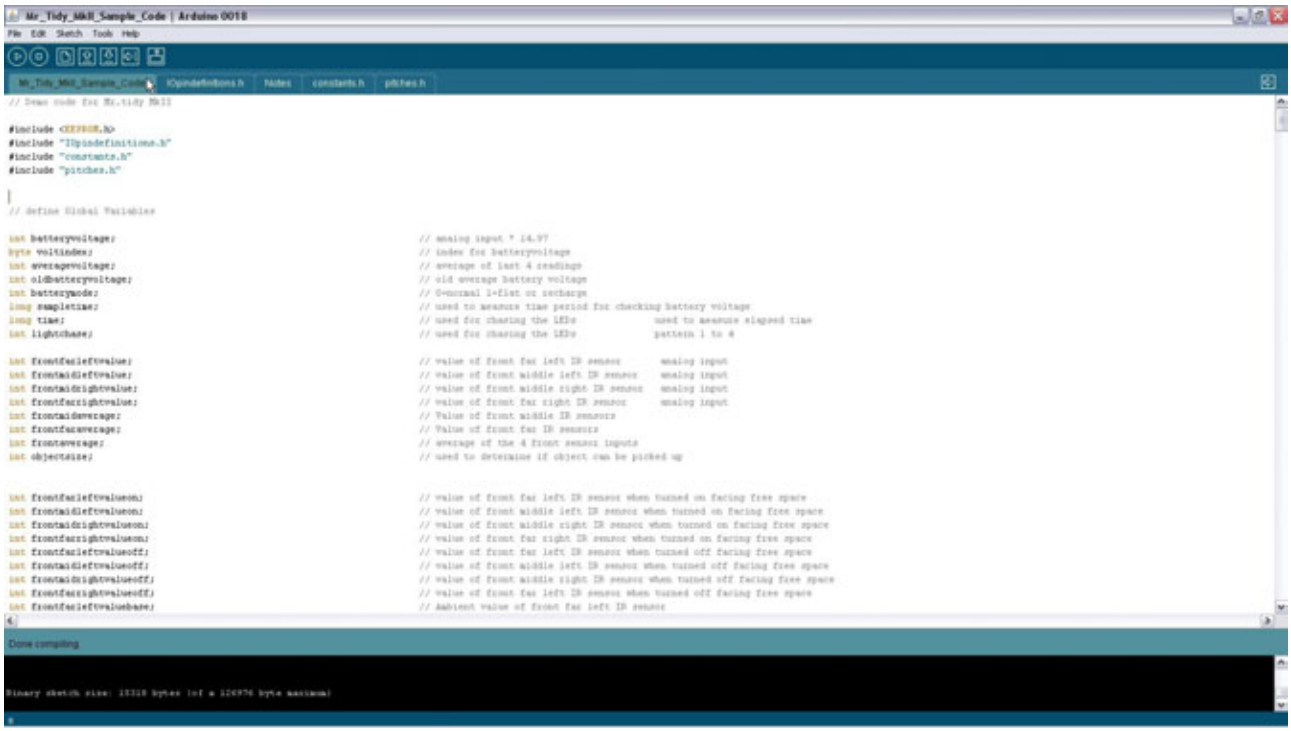
Once you have the Arduino IDE installed on your computer you can then open the program "Mr_Tidy_Sample_Code.pde" which is supplied with the robot or can be downloaded from our website at http://arexx.com.cn/en/DownList.asp



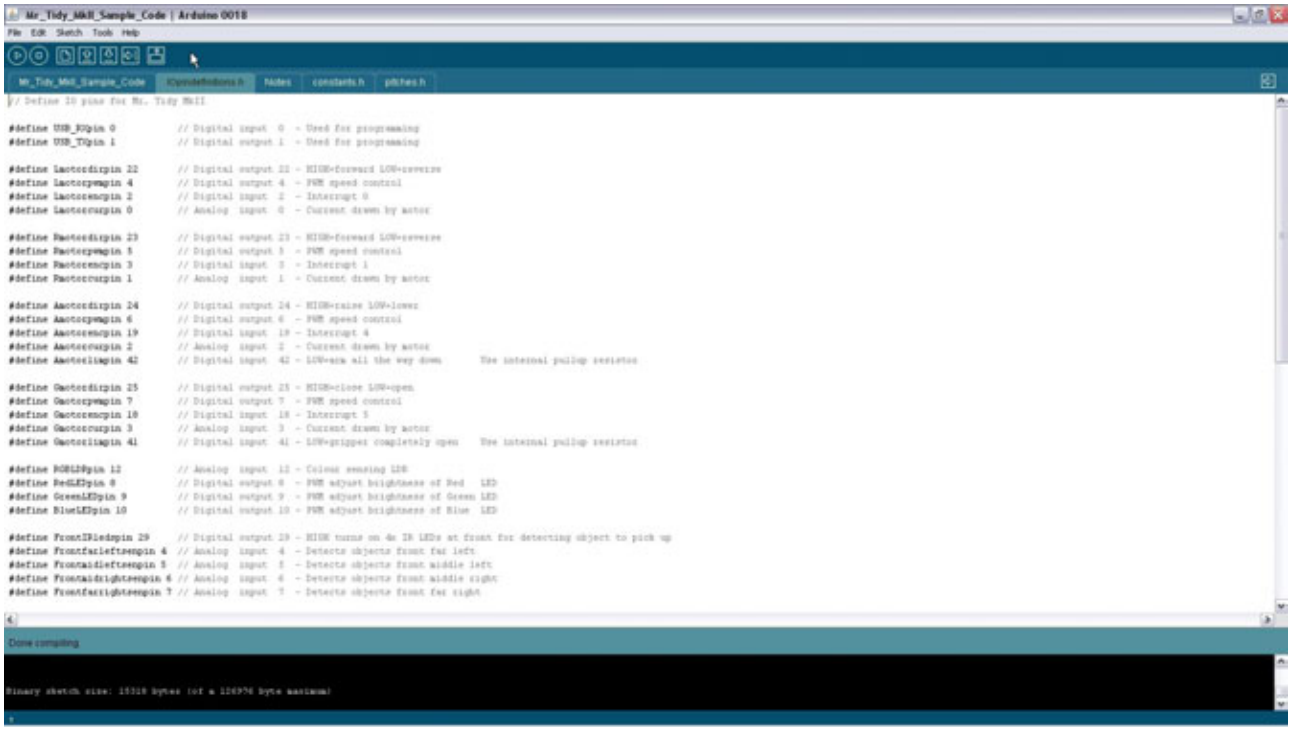
Understanding the sample software

The sample software supplied with the robot instructs the robot to locate objects and sort them based on colour. This is only a simple demonstration program but it contains sample code for all the motors and sensors. The sample code has been written using the Arduino programming environment, as this is easier for beginners.

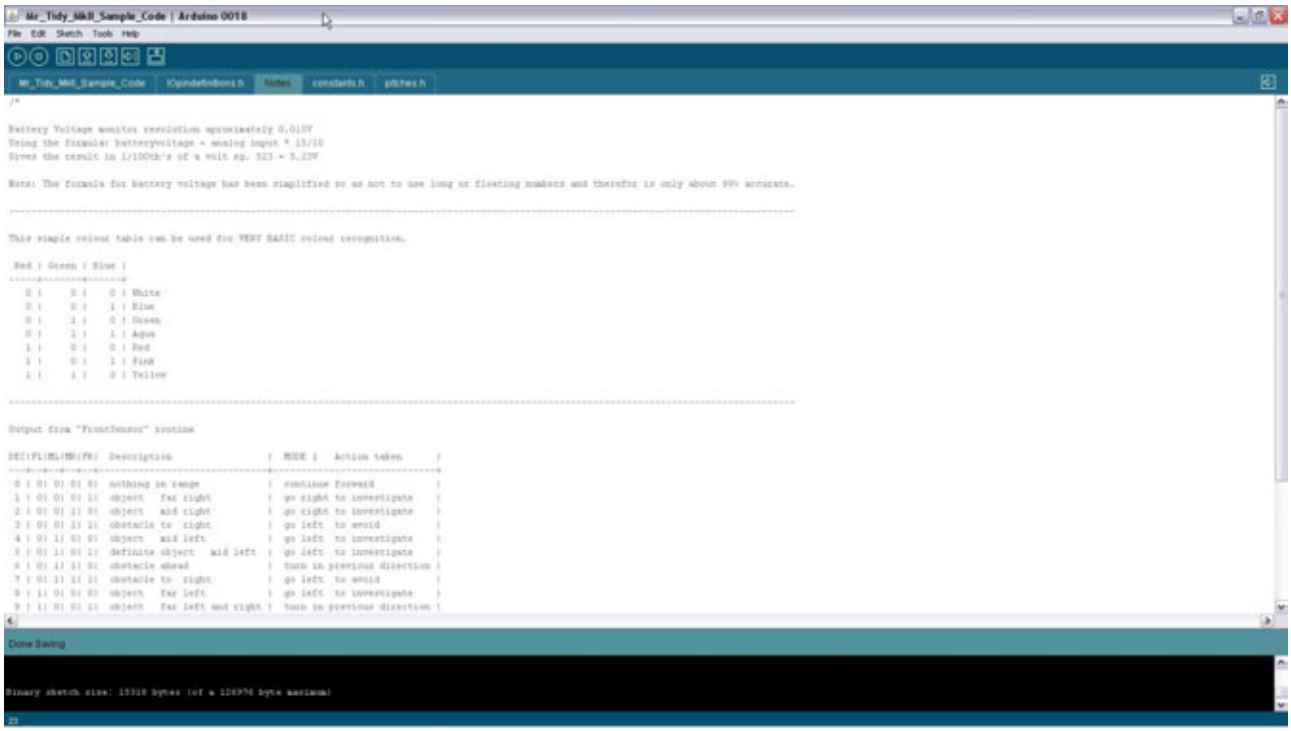
You will see several tabs at the top, these tabs help organise the program making it easier to use. The first tab shows you the main program. You can edit the program here.



The second tab defines the robots IO pins. The Atmega1280 has 70 IO pins of which more than 40 are used for sensors and motor control. If you add additional circuitry then you can define the new IO pins here. The contents of this tab are like a map of the robot.

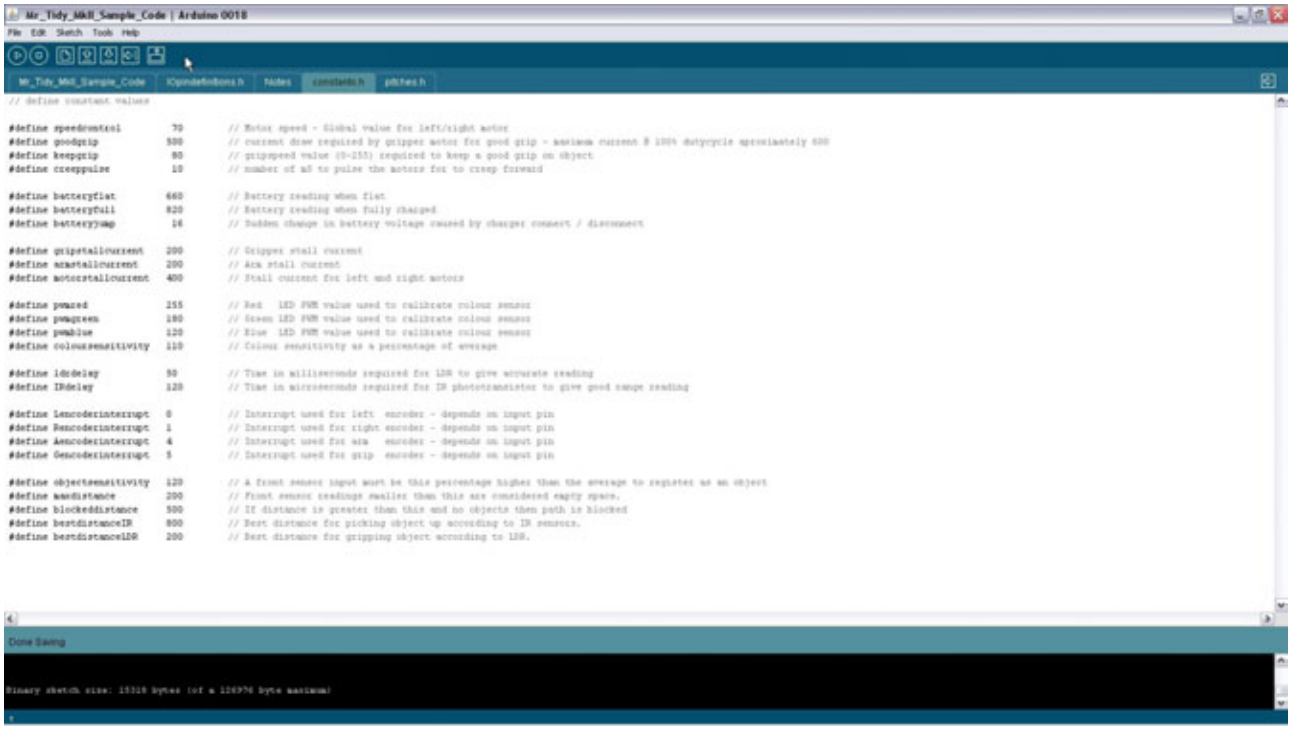


The third tab contains notes on how some of the sensors are used in the software. As you write your own code you may add more information here. This is basically a scribble page.

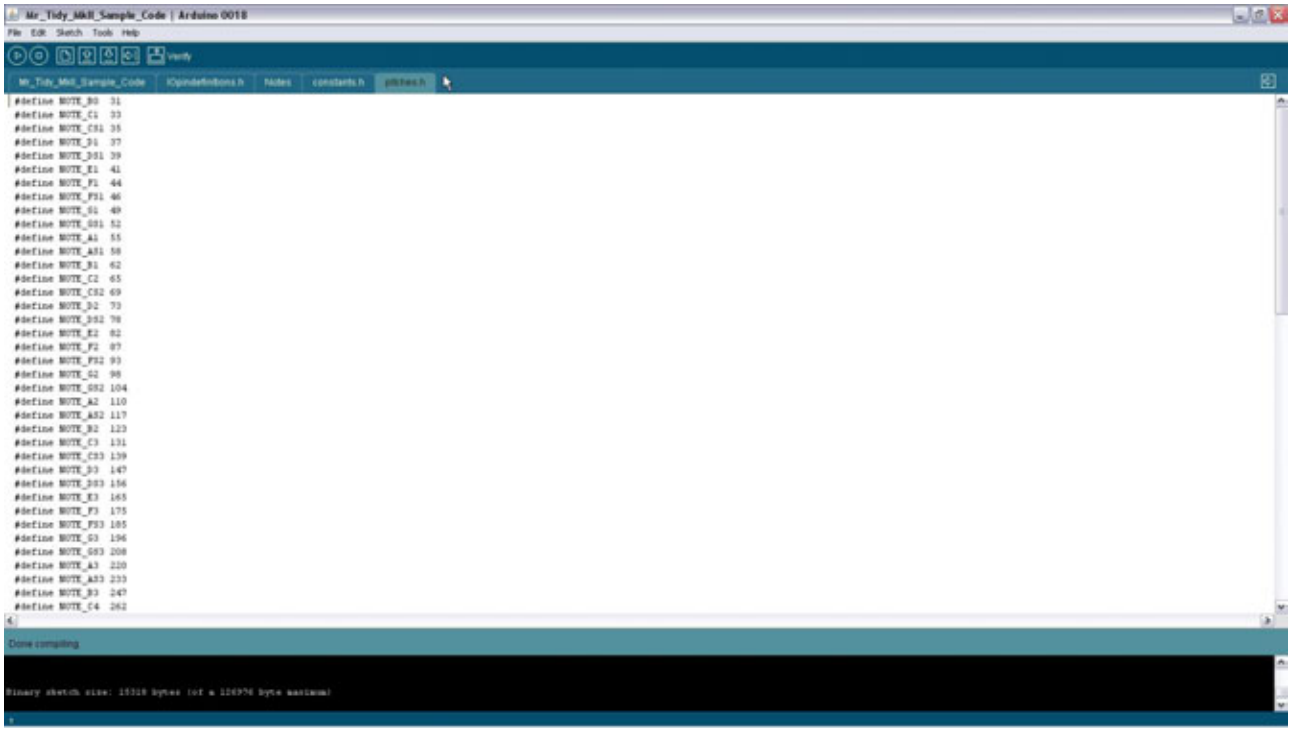


The fourth tab contains a list of constants. Normally these values do not change. They are listed here to simplify program adjustments such as calibration of the RGB sensor.

It is much easier to change a value once here than to search through the entire program changing it where ever it occurs.



The last tab contains a conversion table of musical notes thanks to Brett Hagman. This can be used to simplify generation of melodies for your robot and is used for the initial melody played when the robot begins operation.



Program structure:
When you break the sample program down the structure is very basic. The program starts by including the files that make up the tabs along with any library files required. Global variables are then defined. These variables can be accessed from anywhere in the program.

The **setup()** function as the name suggest is used to setup the robot when the program first runs. The interrupt pins are initialised and output pins are configured. By default, all pins start as input pins when the robot is reset or first powered up. This prevents any accidental short circuits from occurring. The arm and gripper limit pins have an internal 20K pullup resistor initialised.

After playing a simple melody to warn that the robot has activated the gripper and arm are reset to their default positions and the robot is ready to begin operation.

The **loop()** function is the main core of the program and repeats continuously until the robot is shutdown or reset. This function can be broken into several steps:

1. Check the timer and change the LED chase pattern if required.
2. Monitor the **average** battery voltage to determine if the charger has been connected.
3. Shutdown the robot if the battery charger is connected (Press reset after charging is finished).
4. Check the side and rear sensors to prevent a collision and operate/chase LEDs.
5. Monitor the front sensors and create a simple binary value of the readings compared to the average.
6. Perform 1 of 16 actions based on binary readings of the front sensors.
7. Determine if the robot has dropped an object and reset some variables if required.

By performing these seven basic steps repeatedly the robot will locate and pick up objects, attempting to sort objects according to their colour. As this is a simple demonstration program there is no mapping involved with the robot simply choosing a random direction to go in after picking up or setting down an object.

All other functions are called on from the setup() and loop() functions as required. The language reference can be found here: http://arduino.cc/en/Reference/HomePage

How it works

Before you can write your own programs for Mr. Tidy you need to understand how the sensors work and how the motors can be controlled.

Front sensors:

Each front sensor consists of 1 infrared LED and two infrared phototransistors. The two phototransistors are wired in parallel to increase the range. These are analog sensors. As ambient infrared light will affect their readings the software needs to take two readings.

The first reading is taken with the IR LEDs turned on. This value is a total of both ambient light and light from the IR LEDs reflecting off nearby objects. The brighter the light, the higher the value.

The second reading is taken with the IR LEDs turned off. This value is the ambient light only. By subtracting the ambient light value from the total light value you are left with a value equivalent to light reflected by nearby objects.

You will see in the sample software that a small time delay (IRdelay) is used after the IR LEDs change states. This is required to get the best results from the phototransistors. This value is set in the constants tab and can be adjusted if necessary.

Bright sunlight should be avoided as it will result in high ambient light values and reduce the sensitivity of the sensors.

Side and rear sensors:

The side and rear sensors are similar to the front sensors but have only one phototransistor so their range is about half that of the front sensors. These are analog sensors but are connected to digital inputs as we only need to know if an object is within range. Ambient light will affect their range and may even trigger a false detection if it is very bright.

The side and rear sensors have a visible LED wired in parallel with their IR LED. Unlike the front sensors these LEDs are individually controlled allowing the visible LEDs to be used to display patterns or the status of the robot. This does not interfer with object detection which only requires the IR LEDs to be on for very brief amounts of time.

IR receiver:

The IR receiver is a true digital sensor. Unlike the other IR sensors this sensor is designed to detect IR light modulated at 38KHz. It does not respond to the overall intensity of the light so ambient light is ignored. As with the analog sensors, it's range can be reduced by bright sunlight.

The output of this sensor is normally high. When the receiver detects IR light modulated at 38KHz it's output goes low. This sensor is ideal for detecting signals from a universal TV remote or IR navigation beacons. If two or more robots need to communicate with each other then this sensor could receive data from another robot that was modulating it's IR LEDs at 38KHz.

Colour sensing:

This robot uses a Red, Green, Blue (RGB) LED and a Light Dependant Resistor (LDR) to detect colour. Colour sensing is achieved by shining different colours of light onto an object. By measuring how much of each colour is reflected back the objects colour can be determined. In the sample software the voltage across the LDR is measured while shining red, green and then blue light onto the object. The voltage is also measured with the RGB LED off to get an ambient light reading which is subtracted from the red, green and blue readings so that the values represent the reflected light only.

The sample software then compares the red, green and blue values to the average value to provide a simple 3 bit result. This is not the most accurate method and should be refined if precise colour matching is required. As the red, green and blue elements of the RGB LED are driven by PWM outputs the sensor can be easily calibrated.

Motor control:

Each of the 4 motors on the robot are driven by a “H” bridge. This is a circuit commonly used to control DC motors when direction as well as speed needs to be controlled. **F**ield **E**ffect **T**ransistors (FET) are used on this robot as they are more efficient than **B**i-polar **J**unction **T**ransistors (BJT) thus allowing more power to be delivered to the motors. Each “H” bridge includes a current sensing circuit to allow the robot to monitor the current being drawn by a motor and each motor has a simple optical encoder to measure rotation.

Each motor has 2 processor pins dedicated to control and another 2 dedicated to feedback. These pins are defined in the “IOpindefinitions.h” tab. Using the left motor as an example:

Lmotordirpin controls the direction of the left motor. When this pin is HIGH or “1” then the left wheels will run forward. Making this pin LOW or “0” will cause the left wheels of the robot to run in reverse.

Lmotorpwmpin controls the speed of the left motor using **P**ulse **W**idth **M**odulation (PWM). The Arduino IDE uses the command analogWrite() to generate PWM using values from 0 to 255. A value of 0 will cause the motor to stop completely. A value of 255 will run the motor at full power.

At low speeds the motor may stall. A simple method of preventing the motor from stalling at low speed is alternate it’s speed between the speed you want and a low value that will not stall. For example, if you want a speed of 23 and you know the motor will not stall at speeds of 60 and higher then write your program so that at speeds below 60 the program will alternate the speed between the desired speed and 60.

This will cause the motors power to pulse at a level that can overcome a stall. While the robots movements will not be as smooth it will allow the robot to achieve much lower speeds.

Lmotorencpin is the digital signal from the motors optical encoder. The encoder reads black and white marks on the motor gear train and outputs a “1” when a black mark is read and a “0” when a white mark is read. In the sample software, external interrupts are used to monitor these pins and count when they change states.

As this encoder cannot determine direction the sample software uses the direction the motor is travelling in to determine if it should add or subtract from a value. If the motor needs to change direction then it should be allowed to come to a complete stop first to ensure an accurate count.

Lmotorcurpin is an analog signal that represents the current being drawn by the motor. Using the analogRead() command in the Arduino IDE will give a value of approximately 400 when the motor is drawing 1A. This can be useful not only for determining if the motor has stalled but also if the robot needs maintenance. If for example dust or dirt gets into the gears then the robot may continue to operate but the average current draw of the motor will increase. If the motor draws less current than expected then perhaps the robot has fallen so that the left wheels can spin without load.

Amotorlimpin and **Gmotorlimpin** are the digital input pins for the Arm and Gripper limit switches. The digital inputs are normally held at 1 or HIGH by internal pullup resistors. A low or 0 indicates that a switch is closed.

Using motor feedback:

By monitoring the encoders and current sensors of the robots motors various functions can be achieved. The wheel encoders allow the robot to measure speed and distance travelled allowing the robot to make a map of an area to improve navigation. Current sensing of the wheel motors in conjuction with the encoder information allows the robot to map inclines such as ramps.

Monitoring the current draw of the arm motor when lifting an object can be used to judge the weight of an object and determine if the gripper needs to apply more pressure to hold the object.

Using the arm encoder to determine if the arm is moving or has stalled as the motor current for a heavy object might be similar to the stall current.

The gripper encoder allows the robot to estimate the size of the object and the gripper motors current draw can be used to determine how much pressure is being applied.

Troubleshooting: If the robot fails to operate correctly then please check the list below. In many cases it may be necessary to use a 12DC (500mA or better) power supply plugged into the recharge socket. You may also need to install the diagnostic software which is supplied or can be downloaded from our website at: http://arexx.com.cn/en/DownList.asp

Note: the robot should not be operated in strong sunlight as this can reduce the sensitivity of the IR sensors.

Problem - the power switch is on but the robot does not appear to work.
Solution - check the battery connection. Connect 12V DC supply (rated for at least 500mA) to the charging socket on the robot and press the reset button. With the sample software supplied the robot should play a melody before it starts moving. If you are using your own program then load the diagnostic program and retest the PCB with the 12V DC still connected to the charging socket.

Problem - one or more of the encoders does not work.
Solution - Check the encoder cables are plugged in correctly and that the screws near the encoder are tight. With the gripper and arm encoders, check that the encoder disk is as close as possible to the encoder PCB. Load the diagnostic program and connect 12V DC to the recharging socket. Check the encoder with the diagnostic software and adjust the position of the encoder disk if necessary.

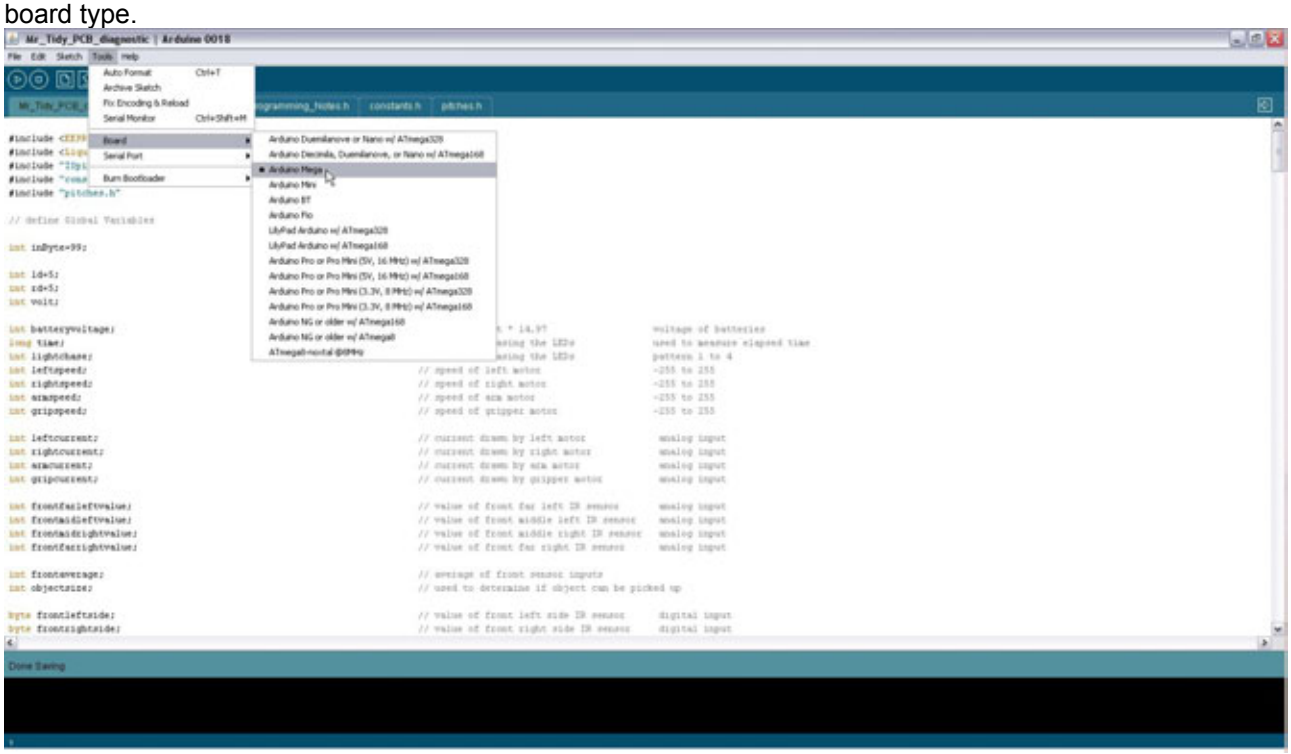
Problem - the robot has trouble detecting objects or aligning the gripper.
Solution - the robot requires the front sensors to be re-aligned. Load the diagnostic program and gently adjust the position of each photo- transistor so that when there are no objects within range all sensors give the same readings (approximately 50).

Problem - one of the collision avoidance sensors does not work.
Solution - the sensor requires re-alignment. Using either the sample software or diagnostic program the blue LED nearest the sensor will stay lit while an object is being detected. The IR LED and phototransistor should be parallel to each other but not touching. Gently adjust their positions until the sensor only detects an object when you place your hand within approximately 40mm of the sensor.

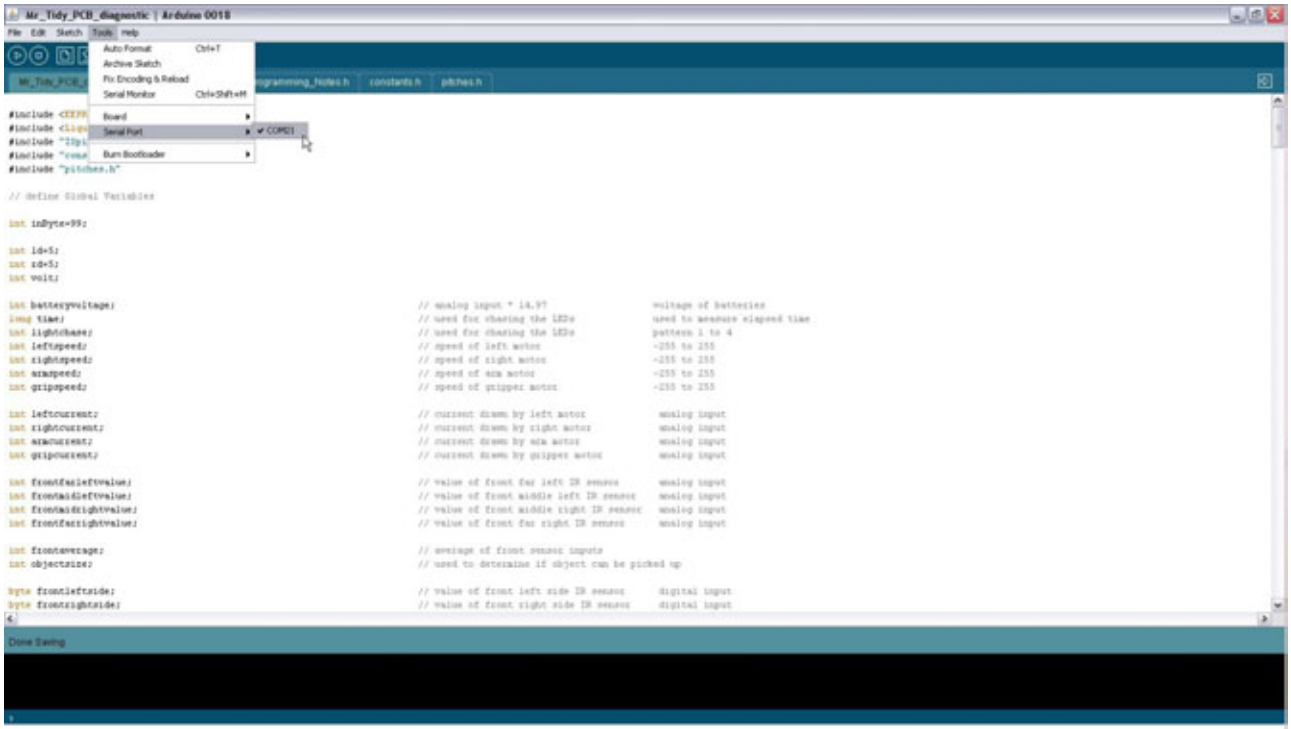
Using the diagnostic program

The diagnostic software allows you to test all of the robots sensors and motors. To use the program you must have the Arduino Programming Environment loaded on your computer (version 18 or later).

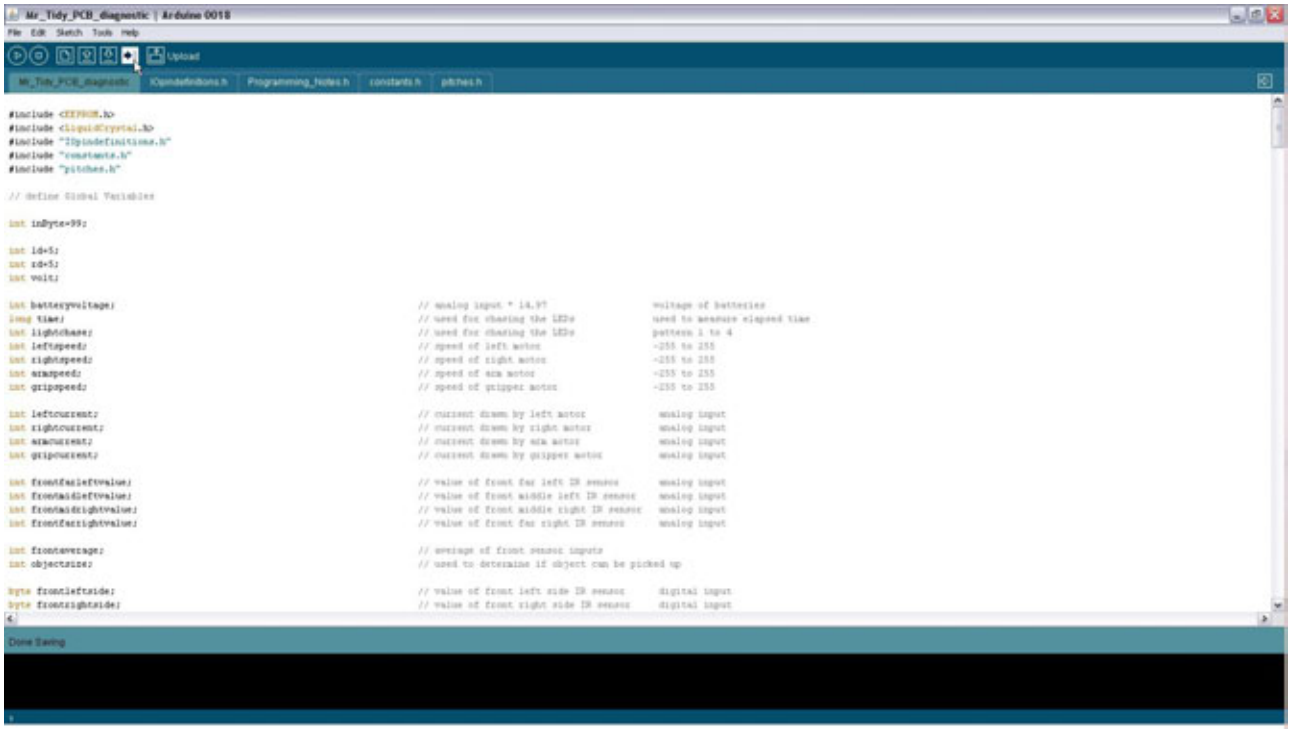
Open the program Mr_Tidy_PCB_diagnostic. Go to the tools menu and select the “Arduino Mega” as your board type.



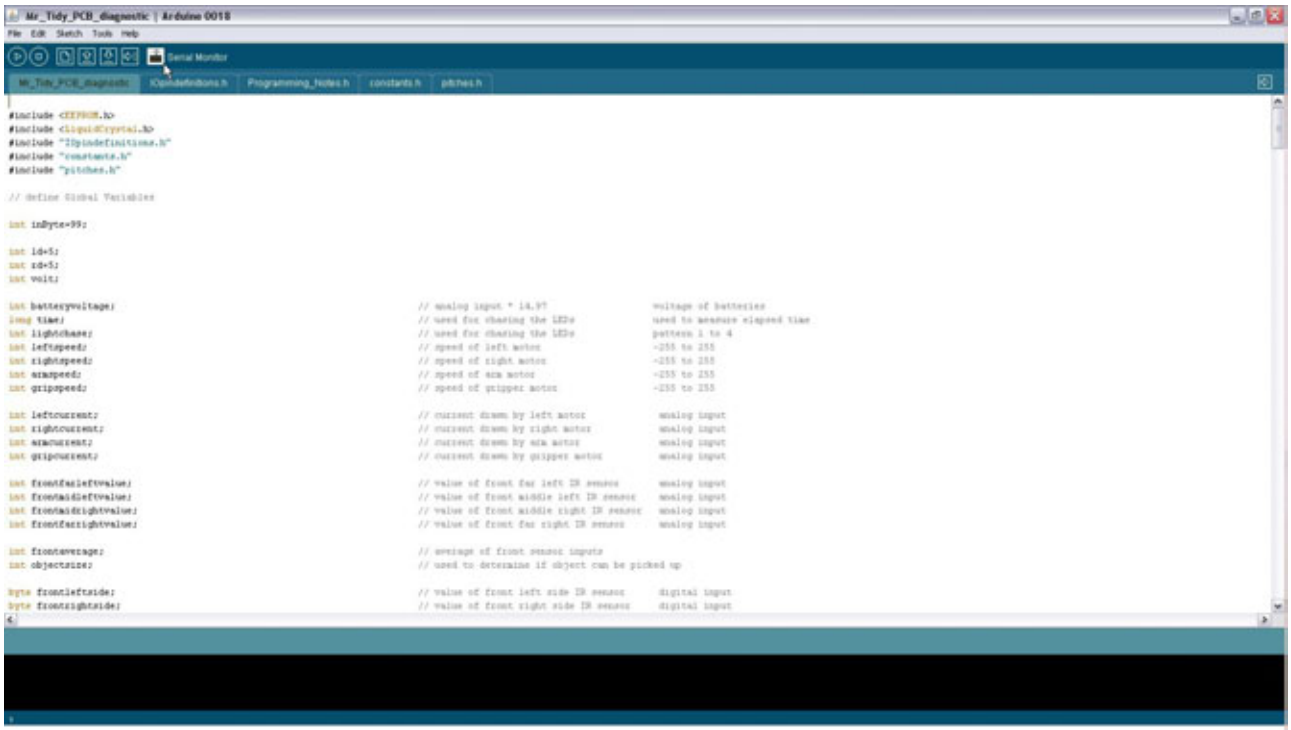
Make sure the robot is connected to the computer via the USB cable and turned on so that the computer can detect it. Select the serial port.



Upload the diagnostic program to your robot. You should see the communication LEDs near the USB port light up as the program is uploaded and verified. If this fails then re-check your serial port settings and check that your robot is turned on. It may be necessary to connect 12V DC to the charging socket if your battery voltage is too low.



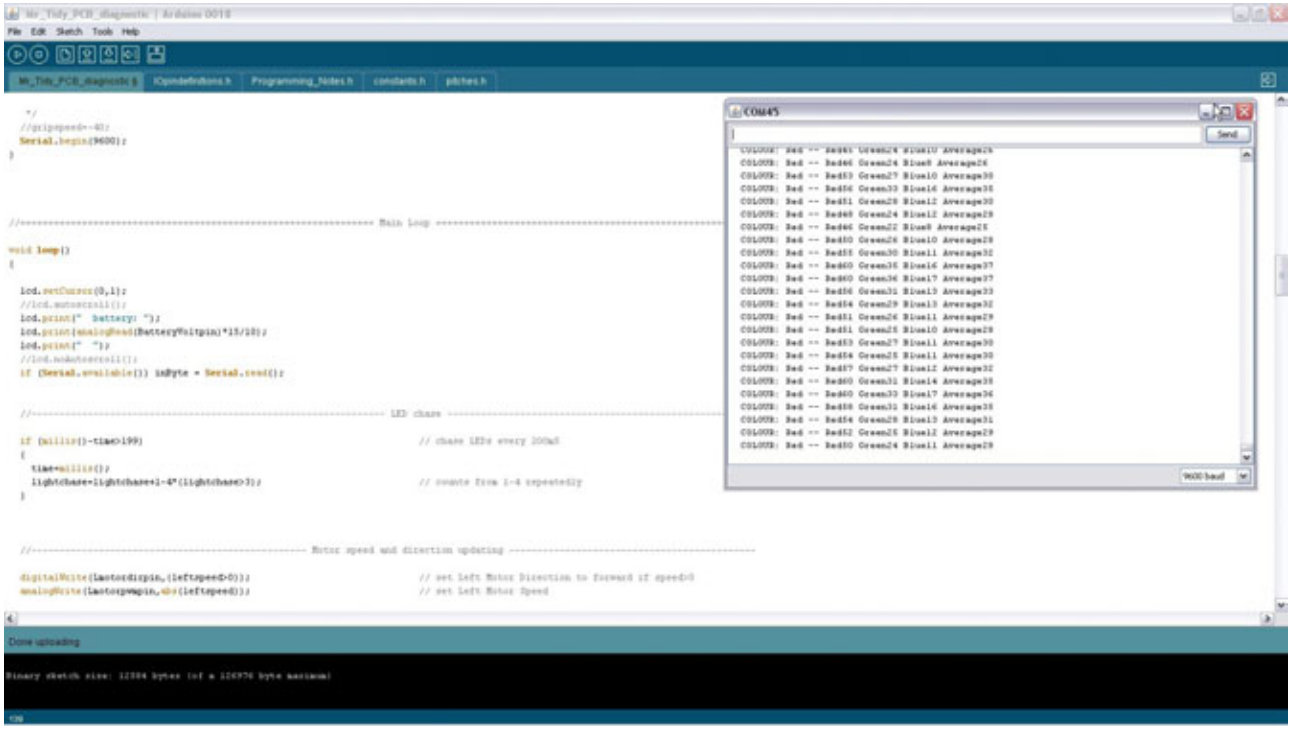
Now run the serial monitor, this allows you to communicate with the robot. Make sure the baud rate in the lower right corner of the serial monitor window is set to 9600.



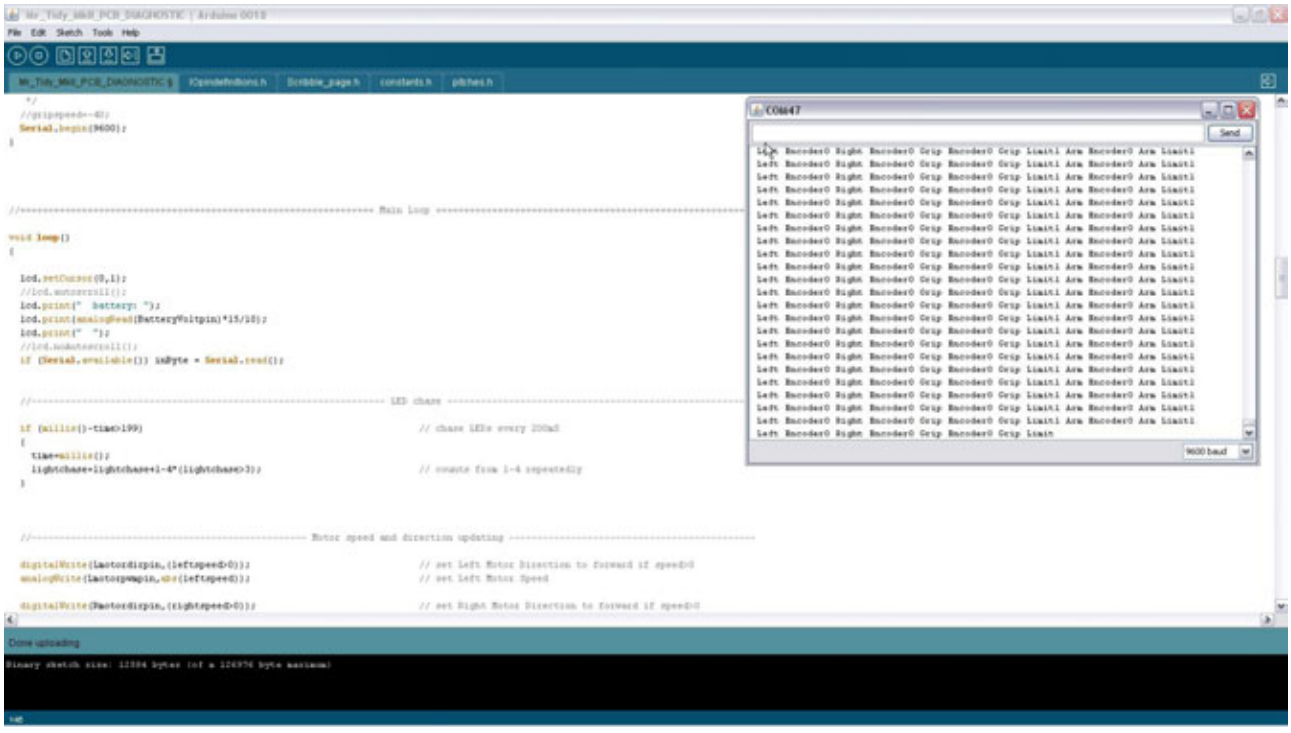
The robot should start by playing a short melody. The blue LEDs at the corners of the PCB should now begin to chase and the RGB LED will flash. In this default mode the collision avoidance sensors can be tested by placing your hand in front of the sensors. When a sensor detects your hand the corresponding LED will stay lit while the others continue to chase.

You will see the output of the colour sensor in the serial monitor. If there are no objects within range it will say the colour is none. The colour recognition code simply looks at which readings are higher than the average.

To calibrate the colour sensor, place a piece of white paper about 20mm in front of the robot. Adjust the red, green and blue PWM values in the constants tab and then upload the program again. When the red, green and blue readings on the serial monitor are the same then the sensor is calibrated. Write these values down so they can be inserted into the main program when it is reloaded back into the robot.



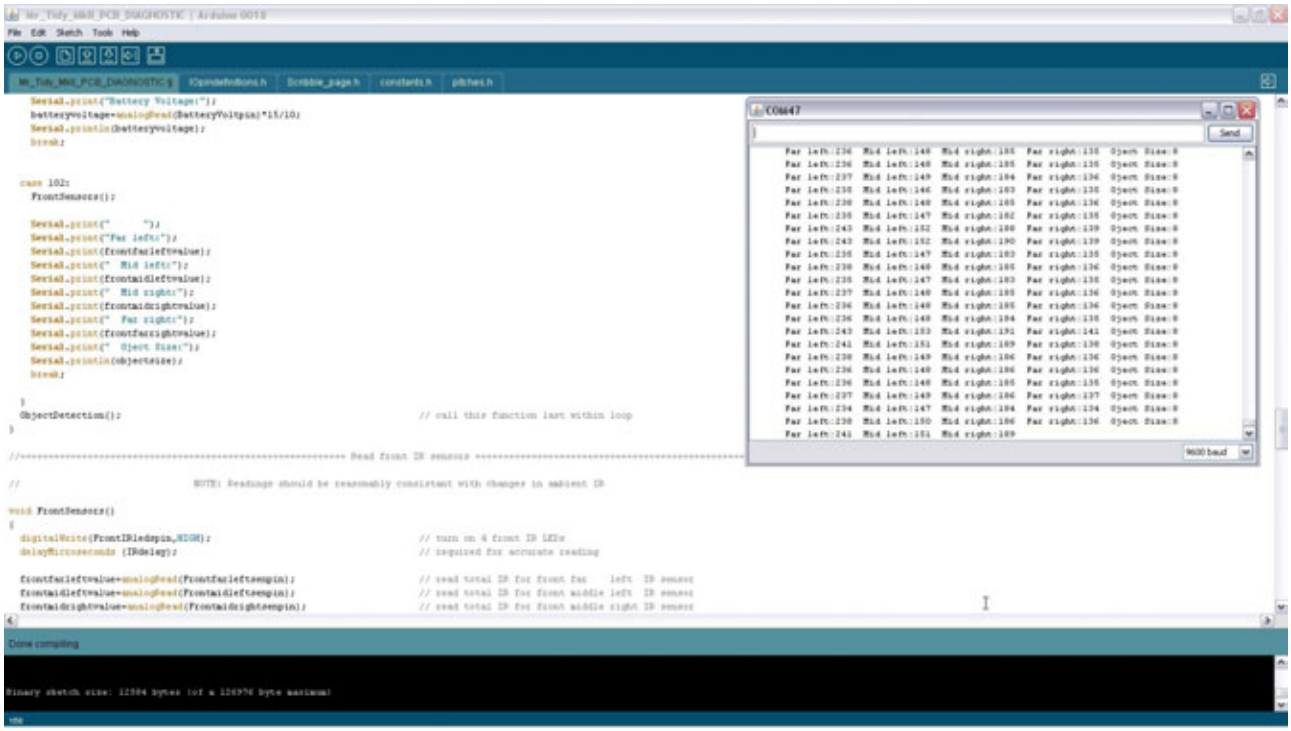
Go to the top of the serial monitor window, type “e” and then press [enter]. This will cause the serial monitor to display the encoder and limit switch inputs.



By manually moving the wheels, arm and gripper you will be able to check the operation of all the encoders and limit switches.

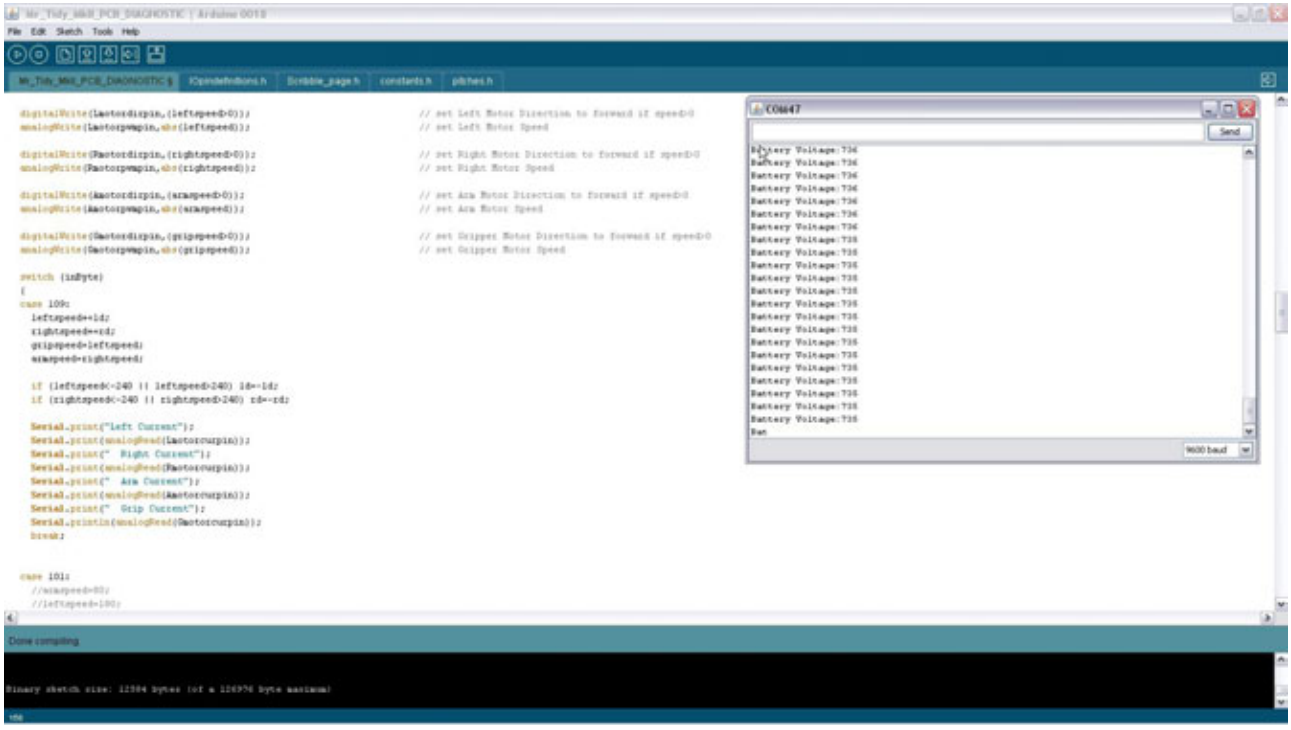
To calibrate the front sensors, press “f” and then [Enter]. This will display the readings from the four front sensors. Place the robot so there is nothing within 1 meter of the sensors. All sensor readings should have similar low values. As you move your hand in front of the sensors you should see the readings change.

The value “Object Size” is the decimal value of the sensor readings used by the sample software to determine what action to take.



The optional sensor alignment rings may affect the sensitivity of the sensors but will help maintain their alignment if the robot bumps into an object.

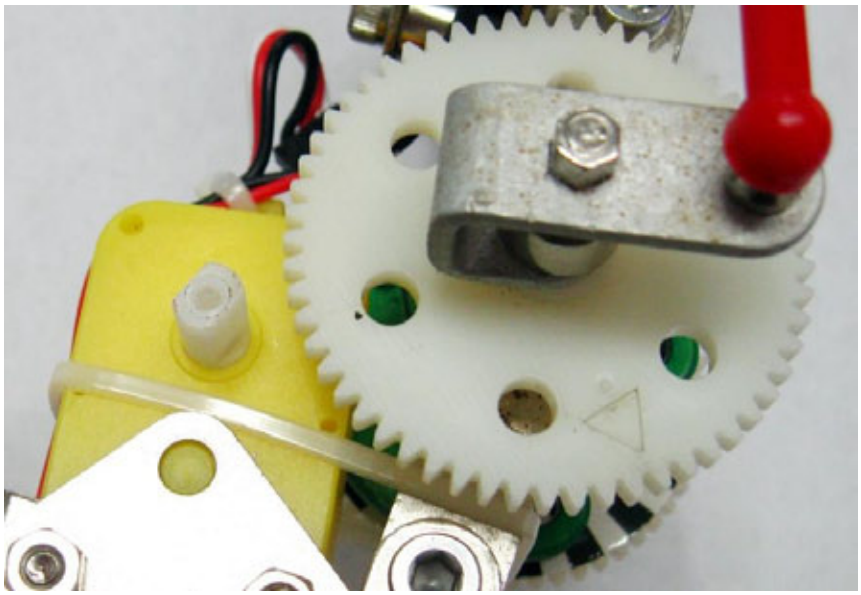
The battery voltage can be monitored by pressing “b” and then [Enter]. The reading is approximately 100x the actual voltage so in this case the battery reading would be 7.35 Volts. This voltage can fluctuate as motors and sensors operate so the sample software averages this over a 20 second period when determining if the batteries are being recharged.



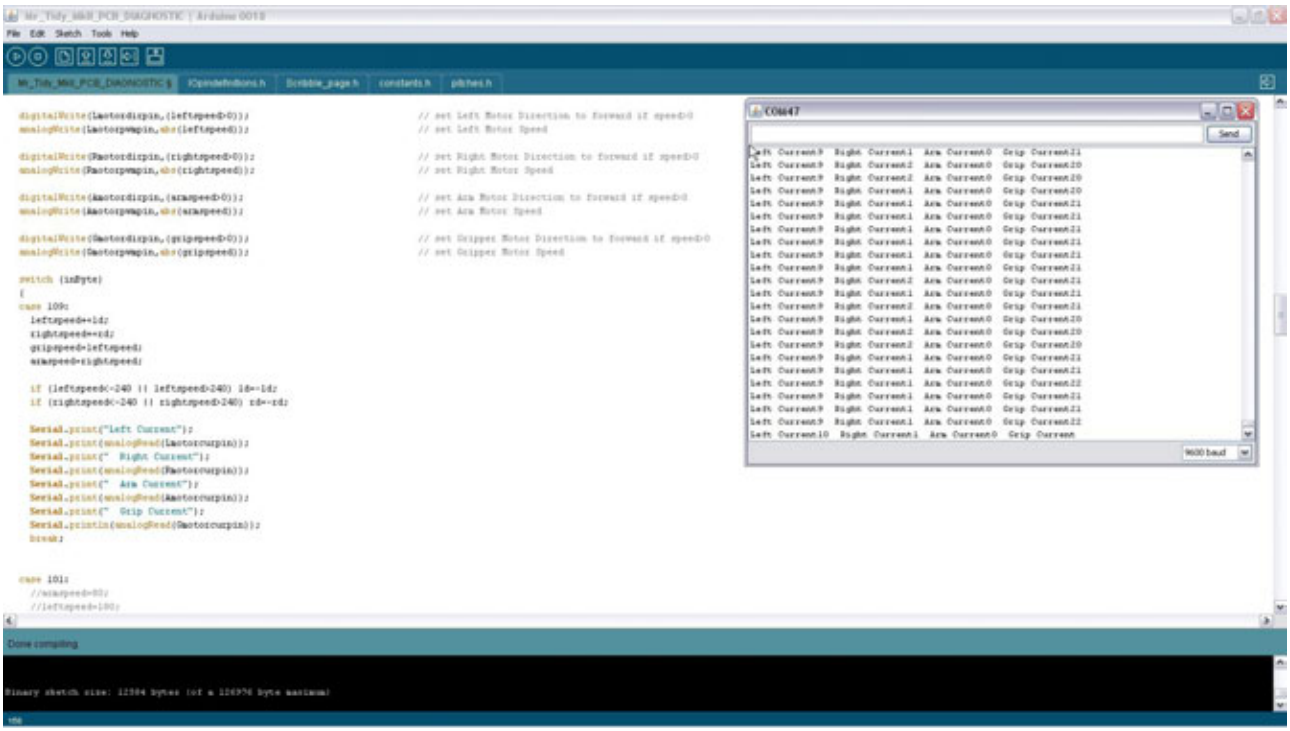
The last test is for the “H” bridges that control the motors. As this test runs all motors forward and backward without regard to limits it is recommended that the arm and gripper motors have their drive gears removed before testing and the robot be suspended so the wheels are not in contact with the ground.

Below you can see a photo of the gripper motor with it’s drive gear removed. Remember the arm and gripper motors have two gears that need to be removed.

If you only need to test one motor then disconnect the other motors from the main board prior to testing.



Once the robot has been prepared press “m” and [Enter] to begin the motor test. As the motors increase and decrease their speed you should see their current draw change.



Specifications

Processor:
Atmega1280-16AU

Clock Speed:
16MHz

Memory:
128K FLASH (2K used by Arduino bootloader), 4K EEPROM, 8K SRAM

Communication methods supported:
USB, I2C, SPI, TTL serial, IR

IR receiver carrier frequency:
38KHz

Motor control:
4x FET “H” bridges rated at 4A continuous with current sensing

Position feedback:
2x Wheel encoder – resolution: 24 counts per revolution
1x Arm encoder – resolution: 27 counts per 90 degrees
1x Grip encoder – resolution: 12 counts from full open to full close

Power:
6x NiMh “AA” or “UM3” NiMh or NiCd rechargeable batteries
DC recharge socket (2.5mm) - requires 12V DC (500mA) to recharge