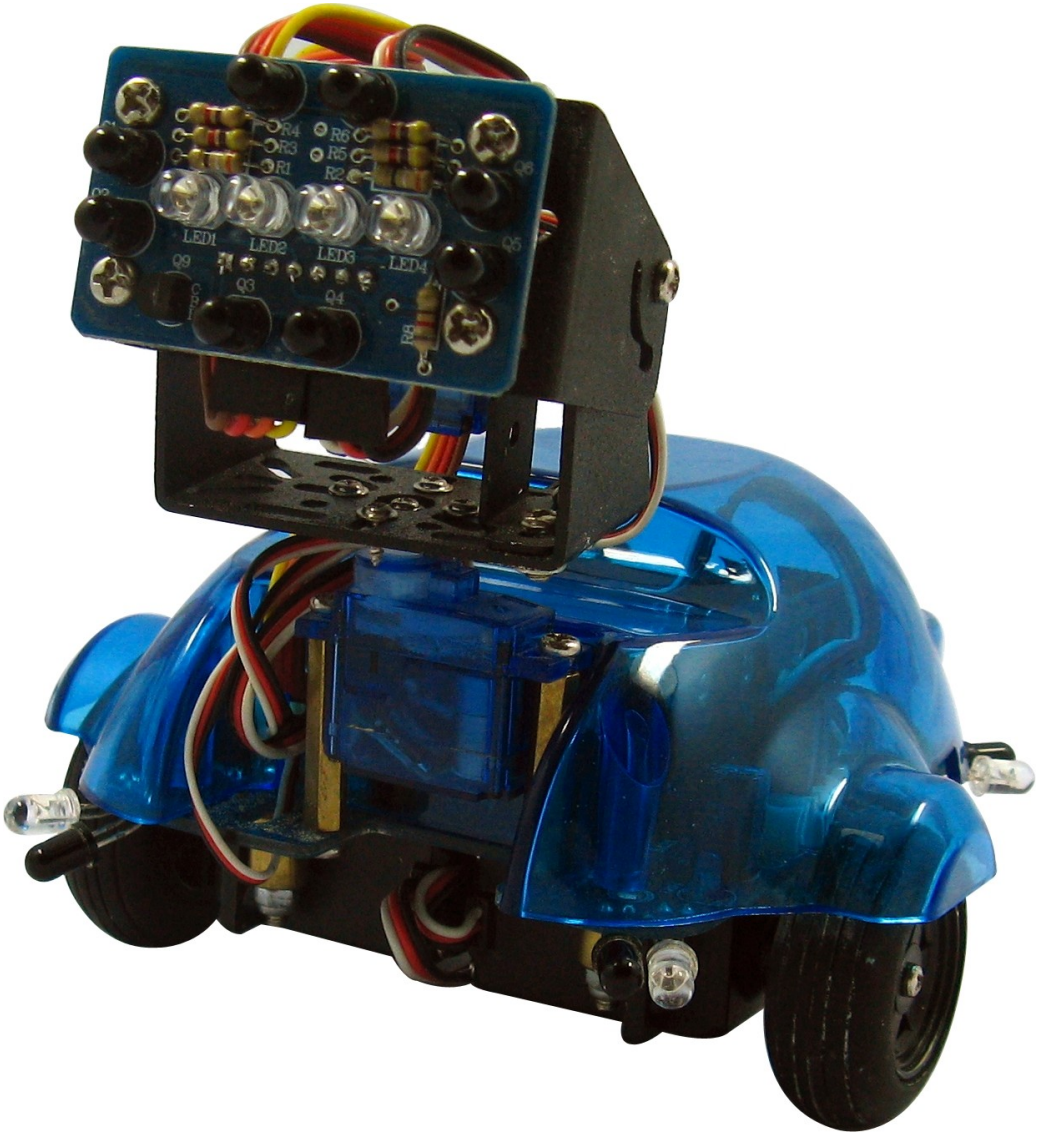
Adventure Bot



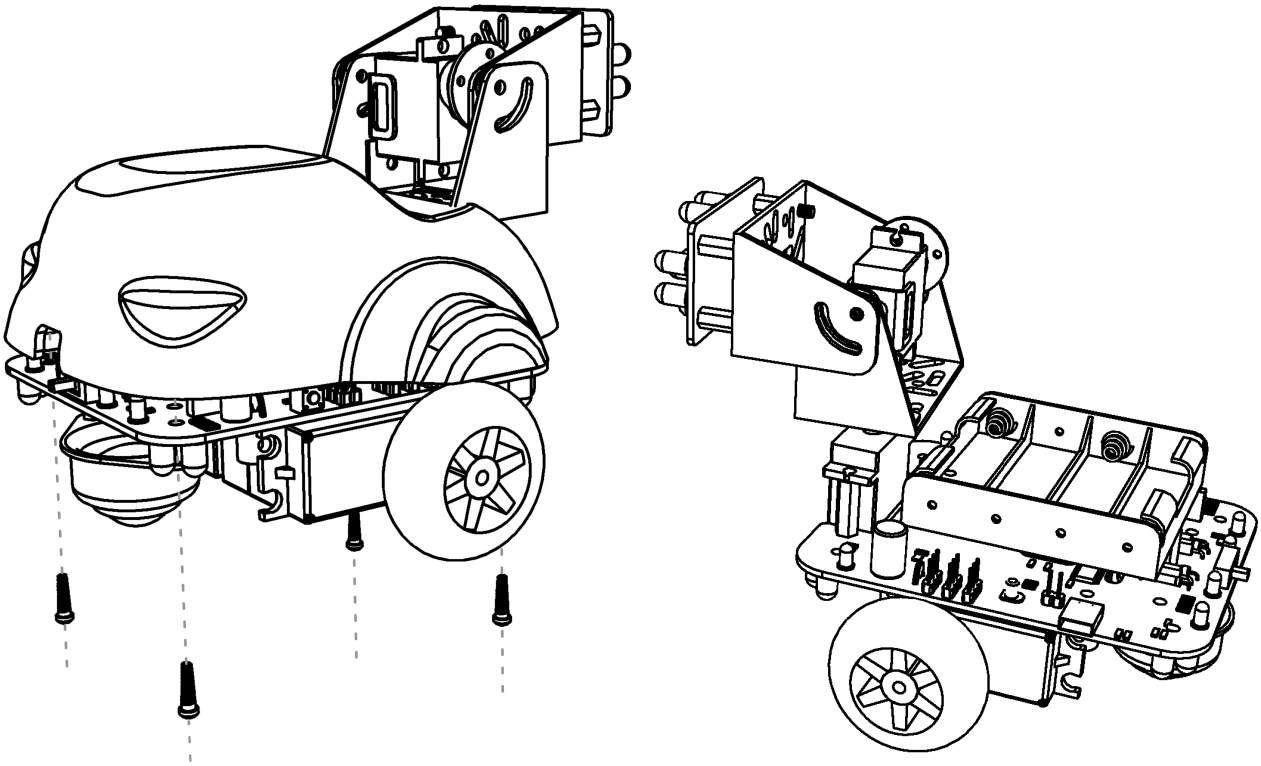
Introduction:
Adventure Bot is a small robot designed to wander around looking for moving objects. Ideal for the beginner or student, Adventure Bot requires no soldering or wiring and comes pre-programmed. Just add 4x NiMh rechargeable batteries. Once the batteries have been installed the robot can be recharged using a standard 9V power pack.

Four corner sensors help the robot avoid collisions. The IR compound eye mounted on a small pan/tilt assembly allows the robot to track the movement of nearby objects and can help the robot judge distance.

The heart of this little robot is an Atmega168 which comes with the Arduino bootloader and demo software pre-installed. The robot is easily reprogrammed using the supplied USB cable and Arduino programming environment or with AVR software and the ISP socket.

Installing the batteries:
Using a small phillips head screw driver, remove the four corner screws holding the cover on. Install 4x AA (UM3) NiMh or NiCd rechargeable batteries. Do not use any other type of battery. Now replace the cover an it's screws.

Recharging your batteries:
When the switch is in the off position it is possible to recharge the batteries by plugging in a 9V DC power pack. This will trickle charge your robot over night. When charging an orange LED near the switch will light up.



Operation:
When you first turn your robot on the red LED at the back should light up and it should play a short tune to indicate that it is working properly. The four corner LEDs should now chase around the robot to indicate that the corner sensors are working. If an object is detected by a corner sensor then it's LED will stay lit until the object is no longer being detected. When you place your hand in front of the robot it will follow the movements of your hand.

Demonstration Software:
The demonstration software that comes with the robot has been written using the Arduino programming environment. The code is a simplified version of the C language ideal for beginners. To modify the demonstration software you must first download and install the Arduino programming environment from this website: http://arduino.cc/en/Main/Software

Once the programming environment is installed in your computer you can download the demonstration software from here: http://www.arexx.com.cn/uploadfiles/Adventure_Robot.zip
Unzip the folder into a directory of your choice and open the file "Adventure_Robot.pde"

Calibration:
When there are no objects in front of the robot the head should return to the center position and the wheels should stop. As no two servos are exactly the same it is often necessary to adjust their center position. Near the start of the program there are four variables that can be adjusted:

```
int neckLRcenter=1490;
int neckUDcenter=1300;
int leftmotorstop=1470;
int rightmotorstop=1455;
```

By adjusting these values you can center the head position and stop the left and right motors.

How the demonstration program works:
The robot has many sensors and motors. If you look at the top of the program you will see a tab called "IO_pins.h". This file tells the program and us which pin is used for what and if the pin is digital or analog, input or output. This is in many ways a map of our robot.

Another tab called "pitches.h" defines musical notes and their equivalent frequencies. This will be very useful when changing the melody played by the sample code. This note table was originally written by Brett Hagman, on whose work the tone() command was based.

At the start of our program we include these files along with the servo library. We also define global variables that can be accessed throughout the program.

void **setup**()
In the setup function we initialise our servos, and configure some pins for output. By default all pins are input (high impedance) when the processor powers up or is reset. Our robot then plays a short melody to let you know it is working ok. If the melody does not play then check that a small jumper near the speaker has been installed. This jumper disconnects the speaker when the ISP socket is used for programming.

void **loop**()
The loop function is the main program and repeats or "loops" continuously until the power is turned off or a reset occurs. Our loop is very simple.

The first section monitors the processors internal clock using the millis() function and changes the pattern of the 4 corner LEDs every 250mS (1000ms = 1 second). This is mainly for effect but does help when diagnosing faults.

The second section updates the 4 servos. The left and right motors are simply servos that have been modified to rotate continuously. Instead of tell these servos to move to a certain position, signals sent to these servos tell them what speed and direction to rotate in. Center position is when they stop rotating. The servos used require a pulse between 0.8mS and 2.2mS sent every 20mS. The writeMicroseconds() function does this for us.

The third section calls functions that monitor the sensors and adjust the speed / position of the servos so that the robot can follow an object without crashing into obstacles.

IReye()
Reads the compound eye once with the IR LEDs on and again with them off. This is important because by subtracting the readings with the LEDs off from the readings with the LEDs on we subtract ambient light from the total light readings to give us reflected light from an object. It also averages these readings to get a distance estimate and plays a note that changes in pitch as this distance changes so that we can hear that the robot is functioning correctly.

IRfollow()
Is a more complex function and has 3 parts. The first part determines if an object is in range using the distance value. If the object is too far away then the robot ignores it and returns it's head to the center position.

The second part of IRfollow moves the pan and tilt servos to allow the eye to follow an object. The position of the pan and tilt servos are adjusted by trying to balance the four inputs (up, down, left, right). Care must be taken to prevent the servos from over correcting without becoming too slow. The variables LRscalefactor and UDscalefactor are used to adjust the sensitivity.

The third part of IRfollow adjust the speed of the left and right motors to help the eye track movement and maintain a certain distance from the object.

ObjectDetection()
This function reads the corner sensors in the same way that the IReye() function reads the eye sensors. It then adjust the motor speeds to try and prevent a collision.

As the corner LEDs used to generate the light pattern around the robot are attached to the corner IR sensors, this function also reads the lightchase variable and leaves LEDs on if an object is too close to that LED or if the chase pattern requires it to be left on.

Where to go from here?
Once you understand the demonstration software it is easy for you to expand upon it.

If desired the corner sensors can be aimed downward to detect sharp drops such as the edge of a table or stairs. If you do this then the ObjectDetection() function needs to be changed.

Add boredom behavior routines so that if the robot does something for too long it gets bored and wanders off.

The IR LEDs and phototransistors around the robot makes "swarm" communications possible. A suitable protocol must be developed so that object detection and communications do not interfere with each other.

Two spare sockets are available on the PCB allowing two additional servos or other devices to be attached (be careful of the power polarity, Vcc is the center pin, Gnd is the pin closest to the outer edge of the PCB and the I/O pin is closest to the center of the PCB.

Have fun and enjoy.