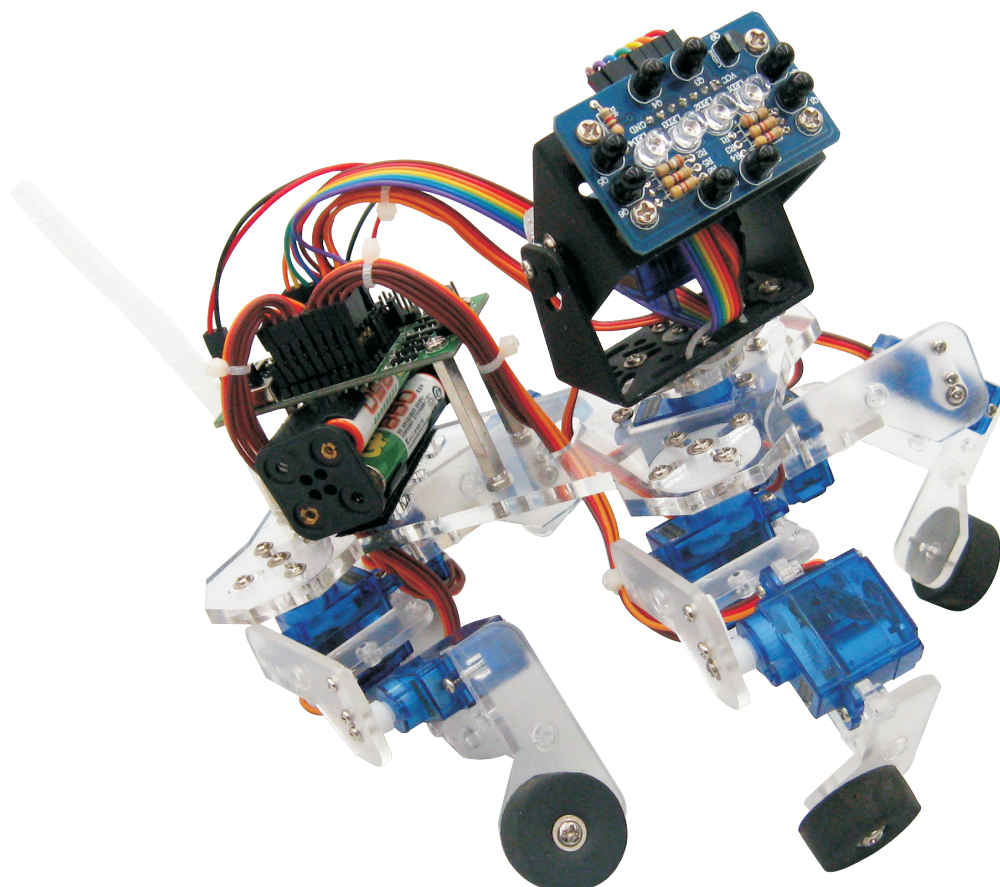


Playful Puppy

Instruction Manual



Copyright©2013 by DAGU Hi-tech Electronic Co., Ltd.

All rights reserved. No portion of this instruction sheet or any artwork contained herein may be reproduced in any shape or form without the express written consent of DAGU Hi-tech Electronic Co., Ltd.

The manufacturer and distributor cannot be held responsible for any damages occurred by mishandling, mounting mistakes or misuse due to non-respect of the instructions contained in this manual.



Manufacturer:
DAGU Hi-Tech Electronic Co.,LTD
WWW.AREXX.COM.CN

Address:NO.4-107/108 HengXing Street, HengHai Rd, South District, ZhongShan City of GuangDong China
TEL:0760-88811951
<http://www.arexx.com.cn>
E-mail:info@arexx.com.cn

CONTENTS

Product description	3
Warnings	3
Required tools	3
Parts list	4
Assembly instructions	5
Wiring Instructions	9
Installing the Software	11
Controlling the robot	13
Understanding the code	14
Uploading via the ISP socket	22
Burning the bootloader	24
Trouble Shooting	25
Controller specifications	26



Product Description

Thank you for selecting the Playful Puppy for your next do-it-yourself project. This kit allows you to build and program a small quadruped robot that can track moving objects and respond to your hand movements.

Product Features

1. Arduino compatible robot controller with USB interface.
2. Laser cut, transparent acrylic base plate.
3. 10x 9g miniature servos.
4. 1x pan tilt assembly with magnetic servo clutches.

Build it now! Realize your dream! Create your next masterpiece!



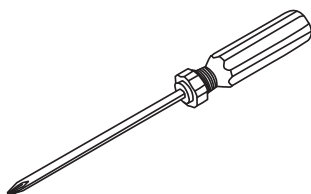
Warnings:

- Opened packages can not be returned. Please check package contents before opening.
- Read the instructions carefully before assembly.
- Use all tools carefully.
- Small parts are a choking hazard. Keep this kit away from young children and babies during construction and operation.
- Not for children under 8 years. Not to be used by children except under adult supervision.
- Observe the correct polarity of the battery.
- Keep the battery dry at all times.
- Do not mix old and new batteries. Do not mix alkaline, standard (carbon-zinc), or rechargeable (nickel-cadmium) batteries.
- Remove the battery if the kit is not used for a long period of time.

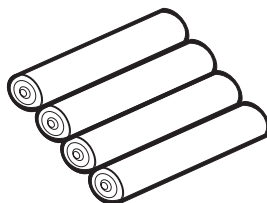
Necessary Tools:

Please read this manual thoroughly before you start assembling the kit. Please follow the assembling instructions exactly to avoid problems. If you work accurately and follow the instructions in this manual exactly, you will quickly assemble your Playful Puppy robot.

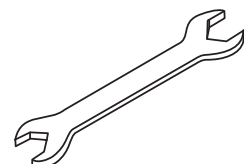
Before you start you must prepare the following tools:



PHILLIPS SCREW DRIVER
(included in kit)

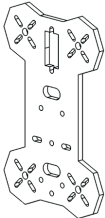
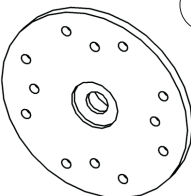
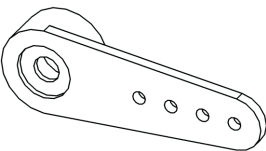
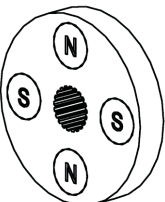
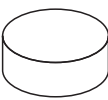
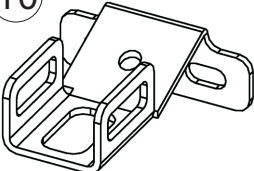
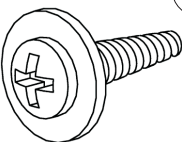
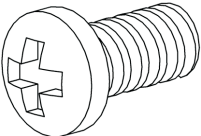
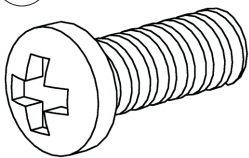
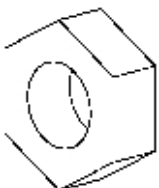
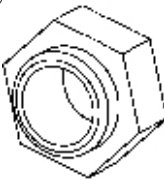
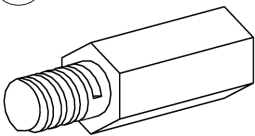
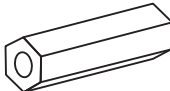
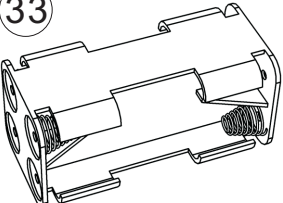
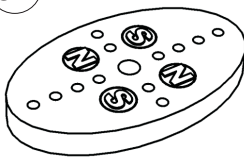
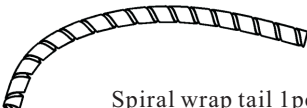


4x AAA BATTERY
(not included)



HEX SPANNER
(included in kit)

Parts List:

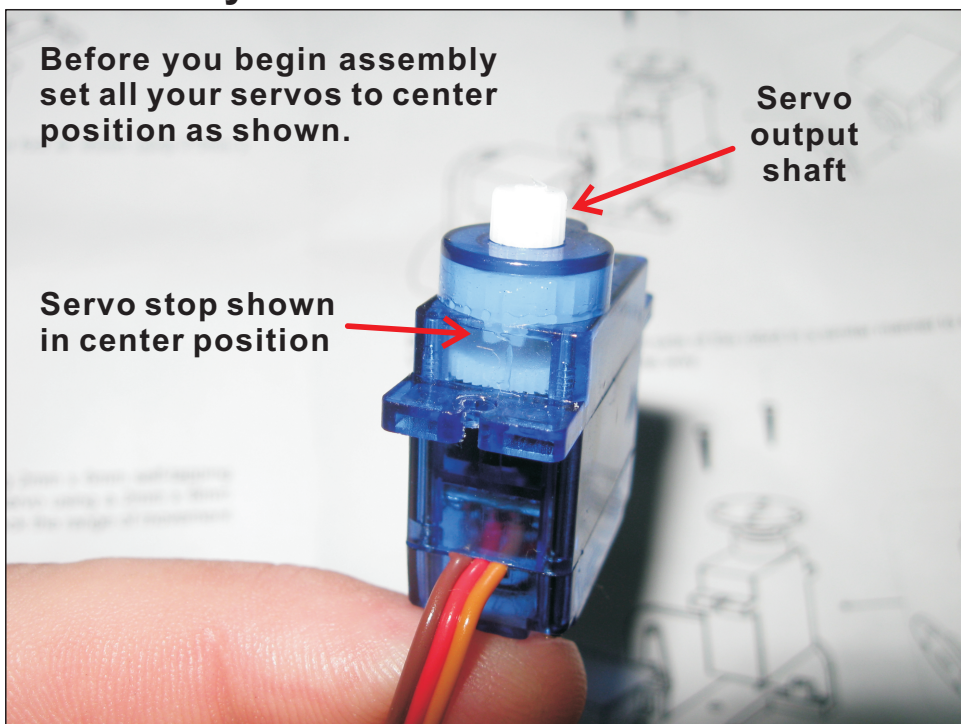
 <p>Base plate 1pc</p>	 <p>9g geared motor 10pcs</p>	 <p>Round servo horn 4pcs</p>	 <p>Servo horn-2 4pcs</p>	 <p>9g servo clutchplate 2pcs</p>
 <p>Clutch-drop shape 1pc</p>	 <p>Leg segment 8pcs</p>	 <p>5x3mm magnet 16pcs</p>	 <p>EVA feet 4pcs</p>	 <p>Servo mounting bracket 1pc</p>
 <p>Pan-tilt bracket-1 1pc</p>	 <p>Pan-tilt bracket-2 1pc</p>	 <p>M2.6*12 self-tapping screw with flange 4pcs</p>	 <p>M2.2*8 self-tapping screw 16pcs</p>	 <p>M2*5 self-tapping screw with flange 6pcs</p>
 <p>M2*6 self-tapping screw with flange 16pcs</p>	 <p>M2*6 self-tapping screw 12pcs</p>	 <p>M2*8 self-tapping screw with flange 4pcs</p>	 <p>M2.3*12 self-tapping screw 2pcs</p>	 <p>M2.6*8 self-tapping screw 2pcs</p>
 <p>M3*6 screw 10pcs</p>	 <p>M3*8 screw 2pcs</p>	 <p>M3 nut 10pcs</p>	 <p>M3 Nyloc nut 1pc</p>	 <p>Washer 5pcs</p>
 <p>L9 brass spacer 4pcs</p>	 <p>L30 brass spacer 4pcs</p>	 <p>Velcro adhesive strips 1 pc male, 1 pc female</p>	 <p>L15 spacer 1pc</p>	 <p>Glassine washer 4pcs</p>
 <p>IR compound eye 1pc</p>	 <p>Mini Driver board 1pc</p>	 <p>Battery holder 1pc</p>	 <p>Clutch-elliptical shape 1pc</p>	 <p>7 core rainbow cable 1pc</p>
 <p>Spiral wrap tail 1pc</p>				

Assembly Instructions:

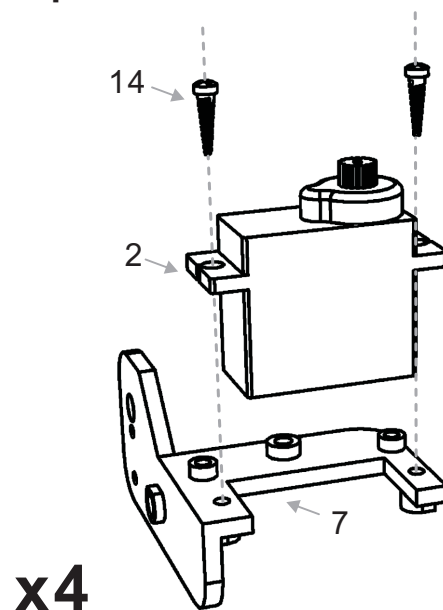
Before you begin assembly set all your servos to center position as shown.

Servo stop shown in center position

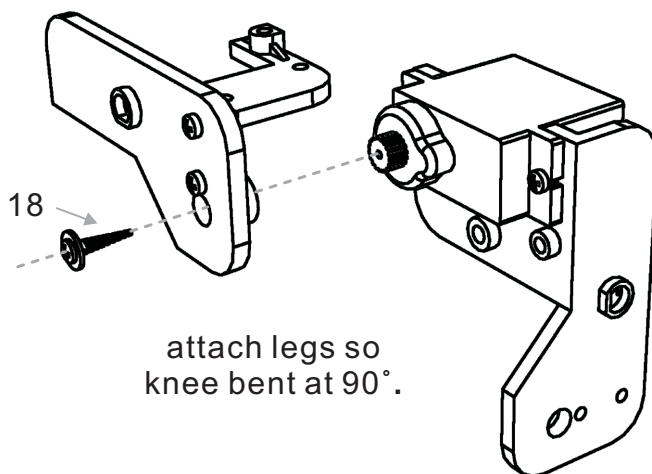
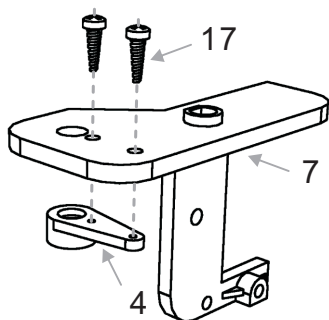
Servo output shaft



Step 1:

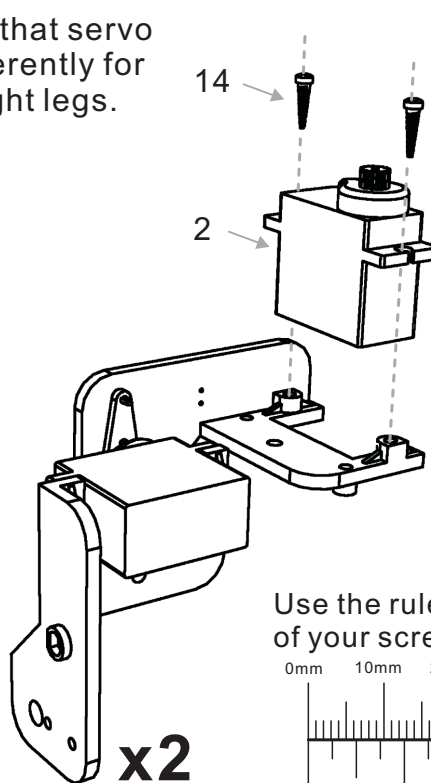
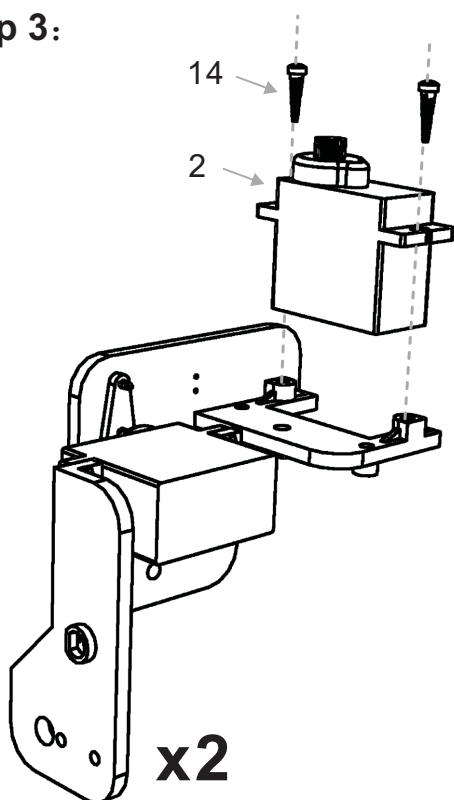


Step 2:

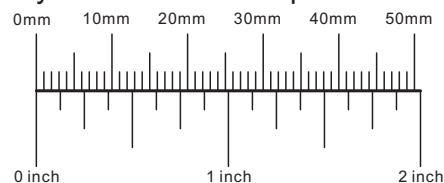


Step 3:

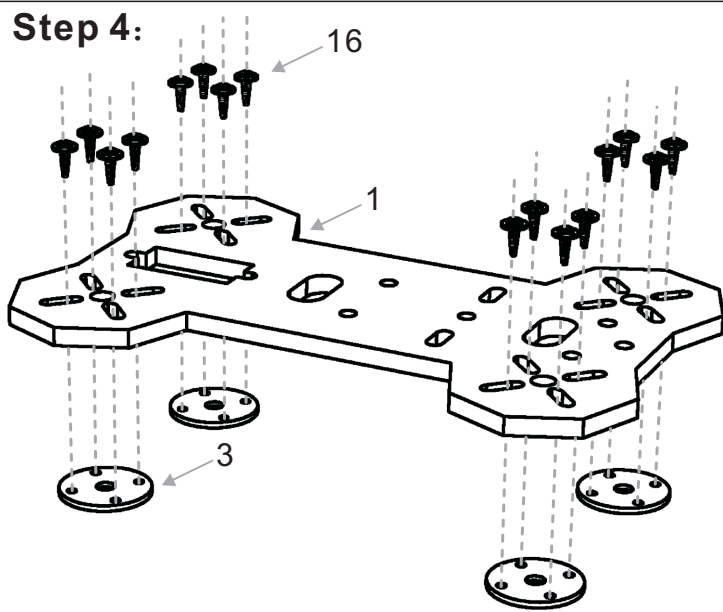
Please note that servo mounts differently for left and right legs.



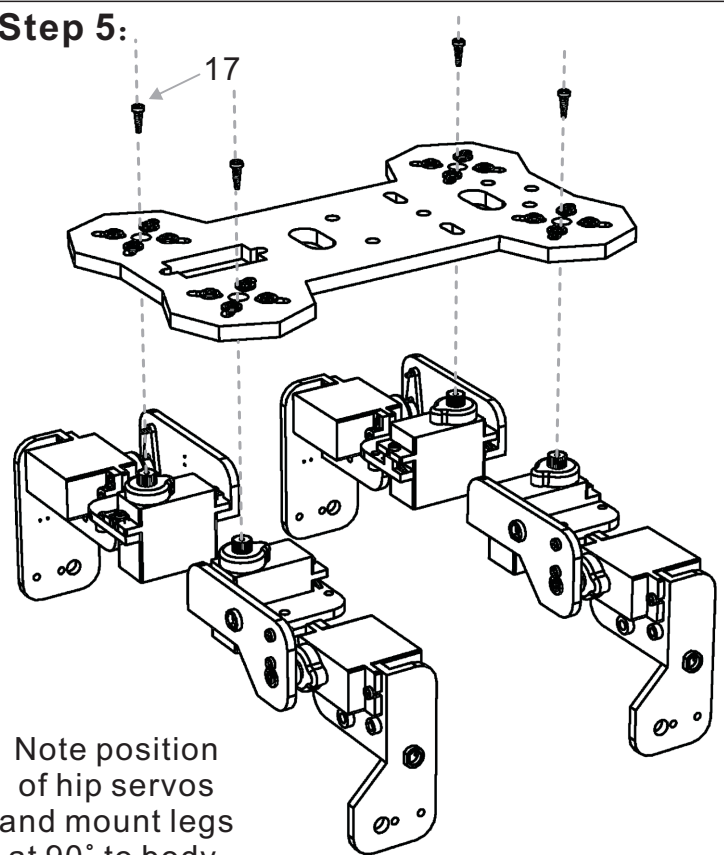
Use the ruler to check the length of your screws and spacers.



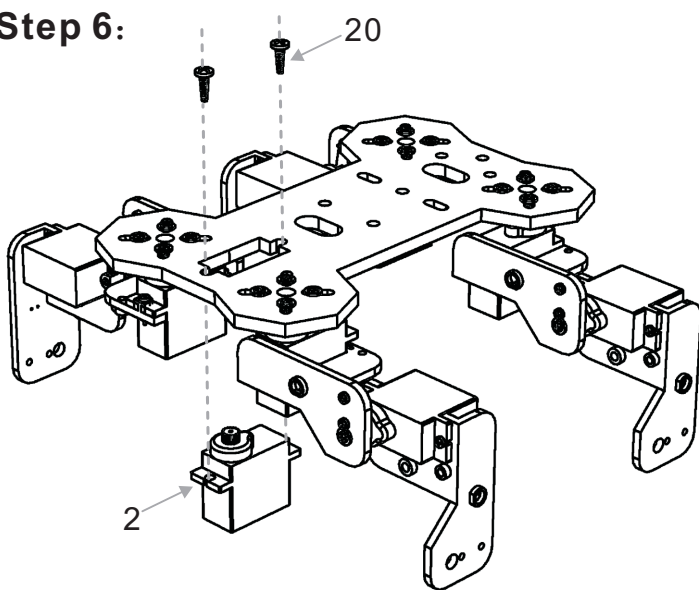
Step 4:



Step 5:

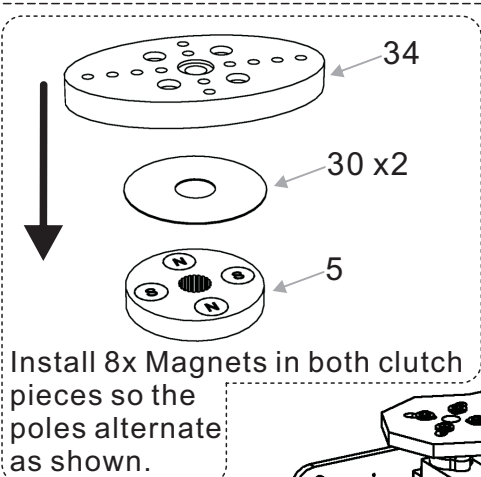
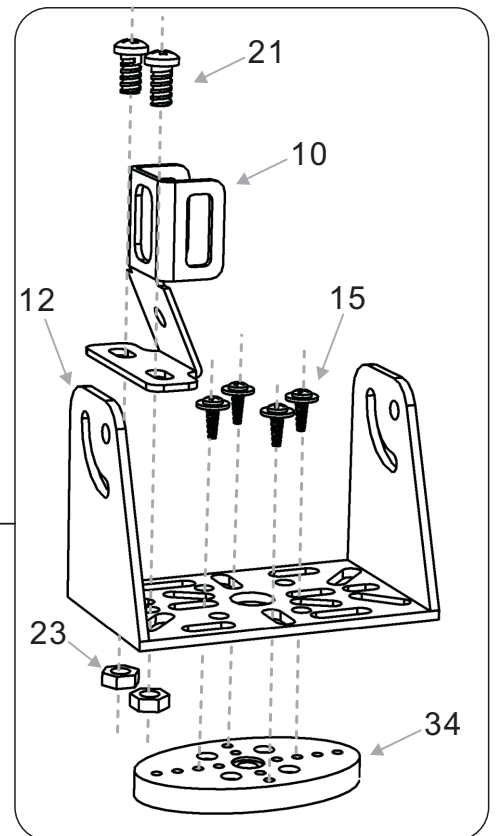


Step 6:

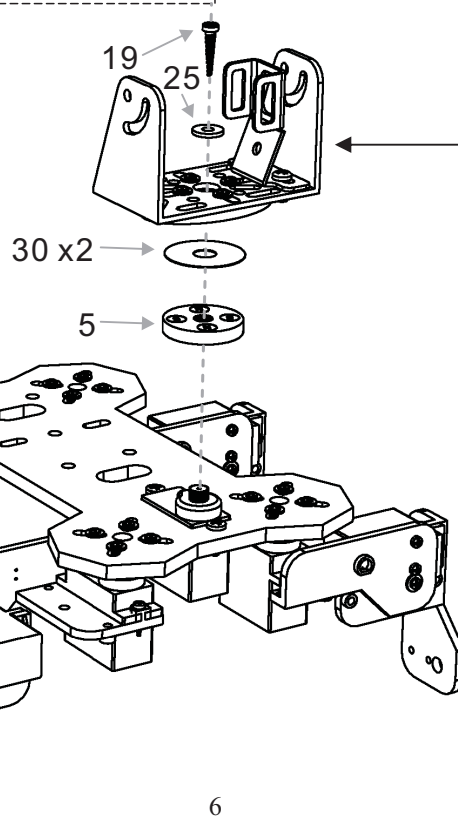


Note position of hip servos and mount legs at 90° to body.

Step 7:



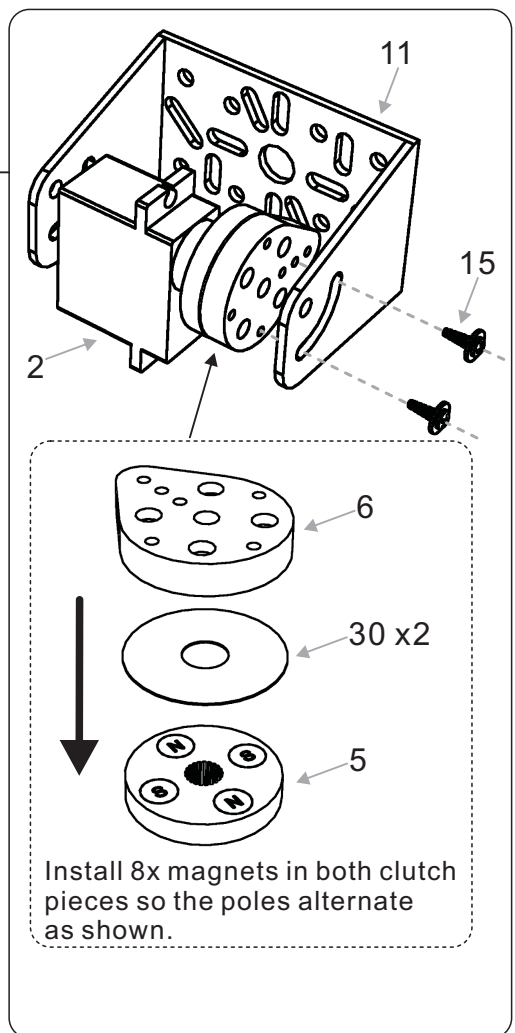
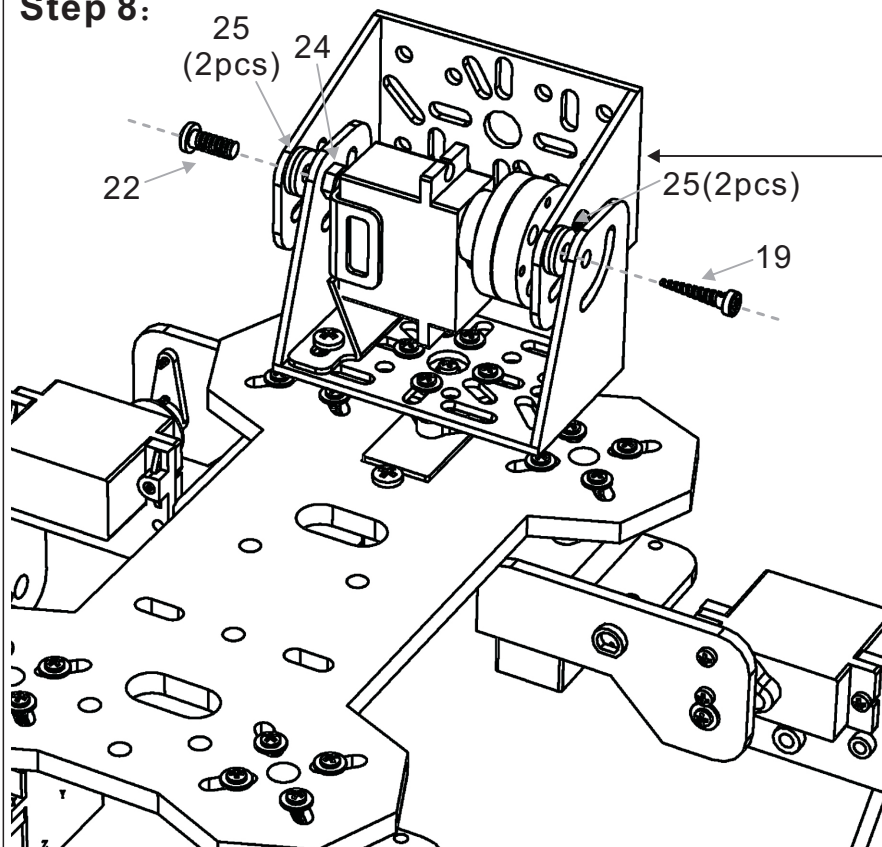
Install 8x Magnets in both clutch pieces so the poles alternate as shown.



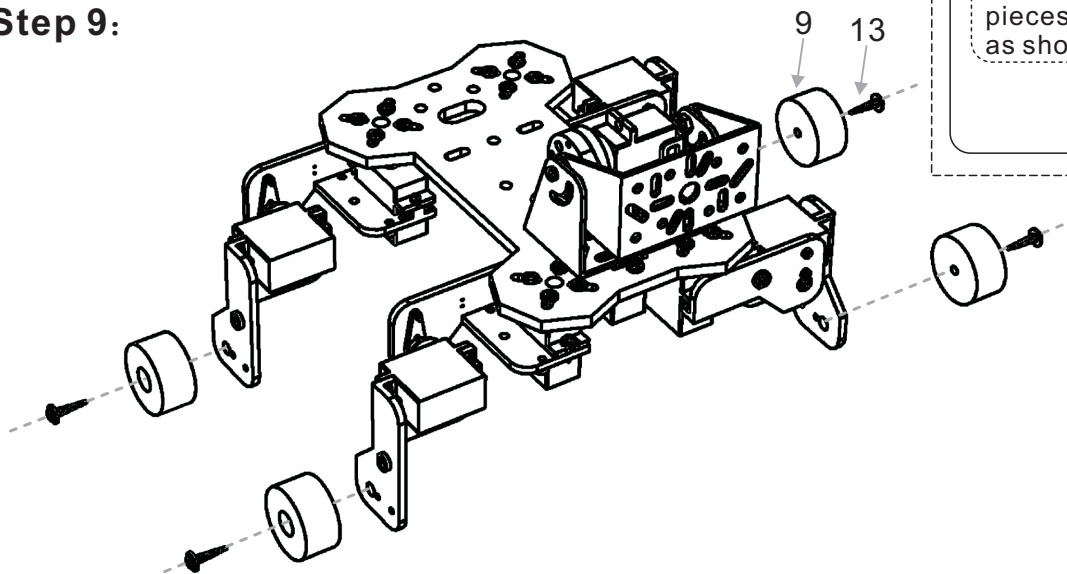
Pay careful attention to which bracket you use. The hole patterns are different to suit various sensors.

Make sure the servo bracket is mounted on the correct side.

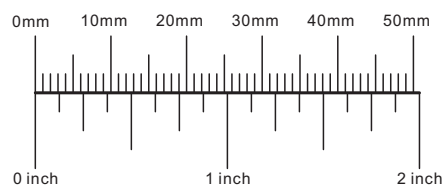
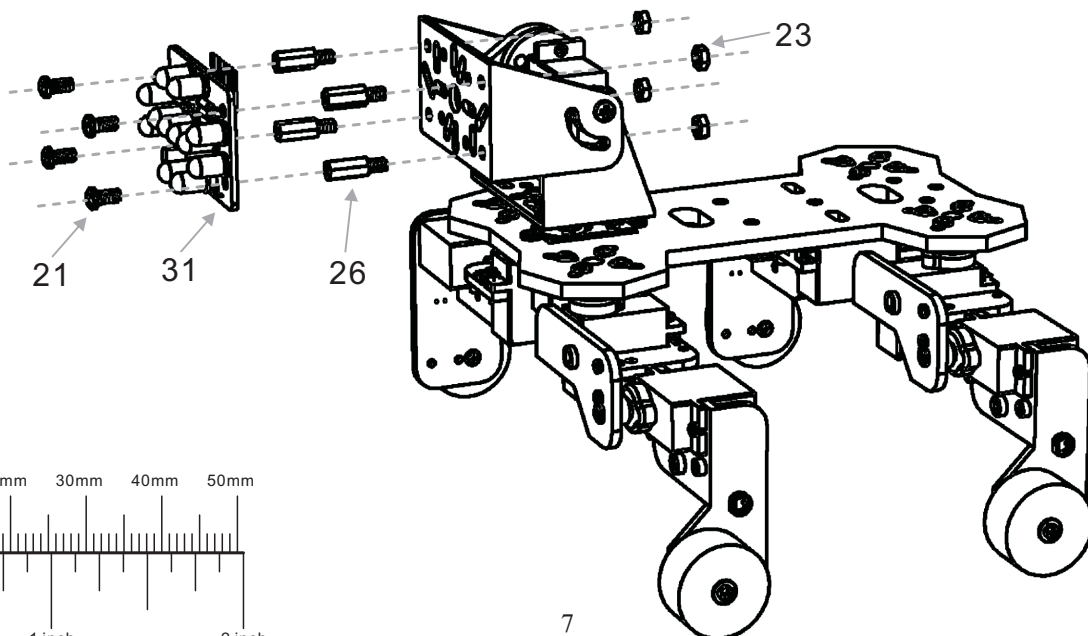
Step 8:



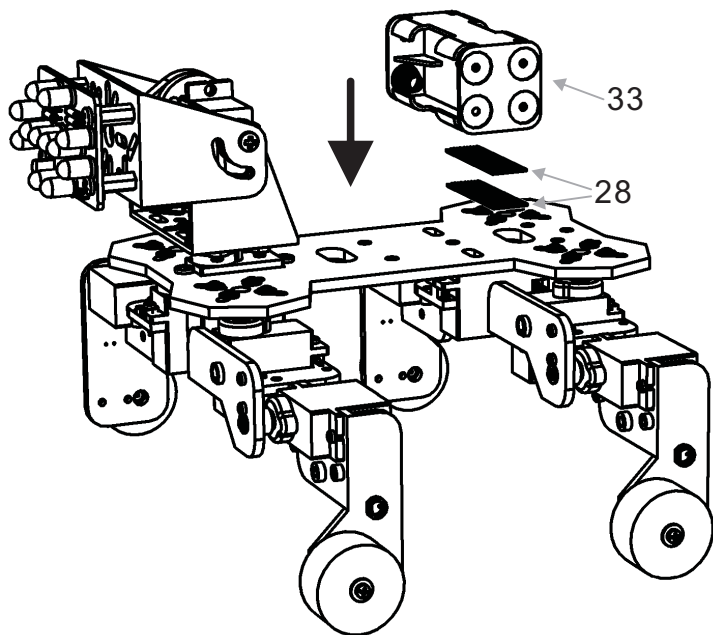
Step 9:



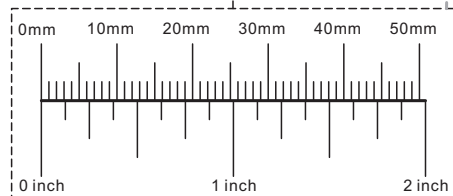
Step 10:



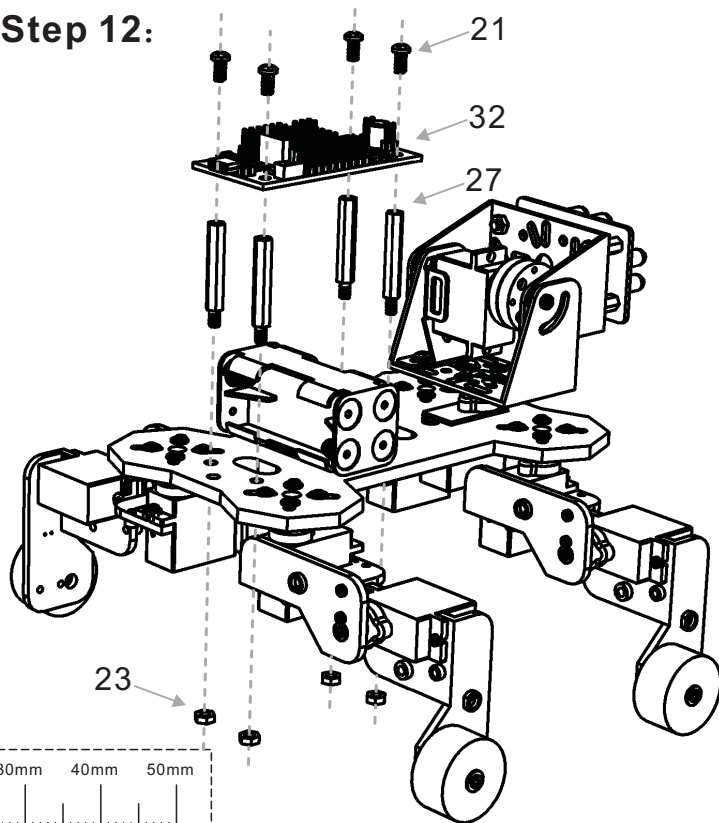
Step 11:



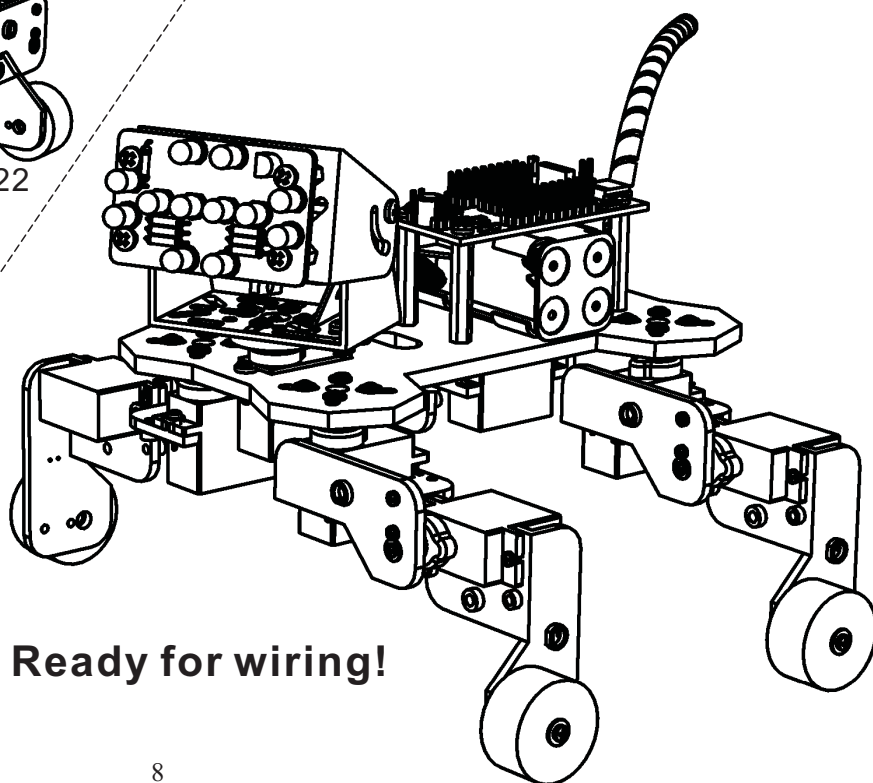
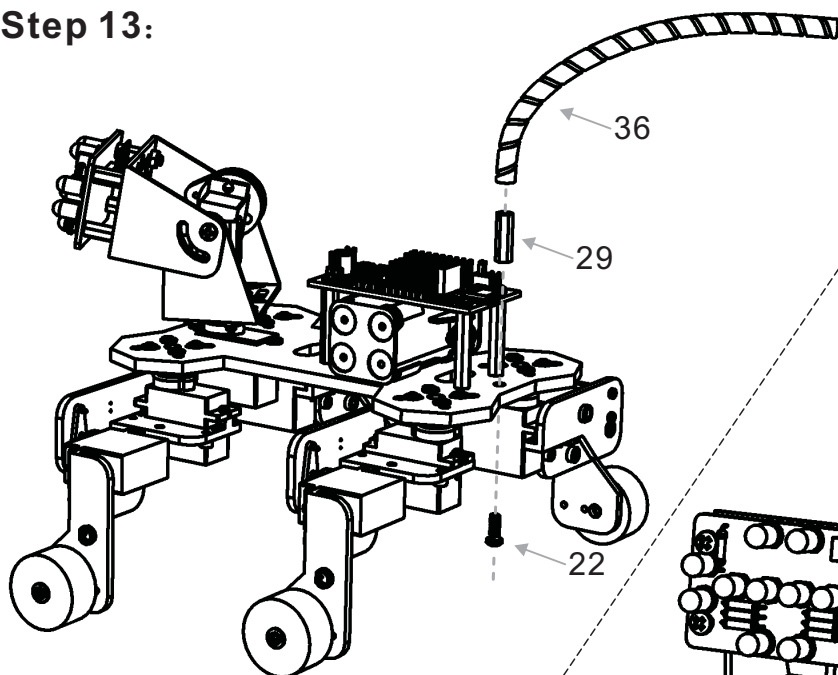
Remove protective paper from adhesive strips and mount battery holder with velcro.



Step 12:



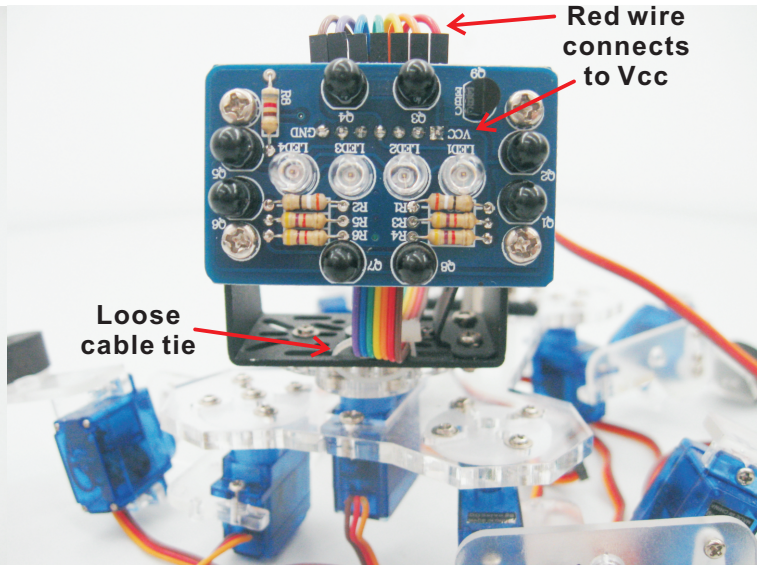
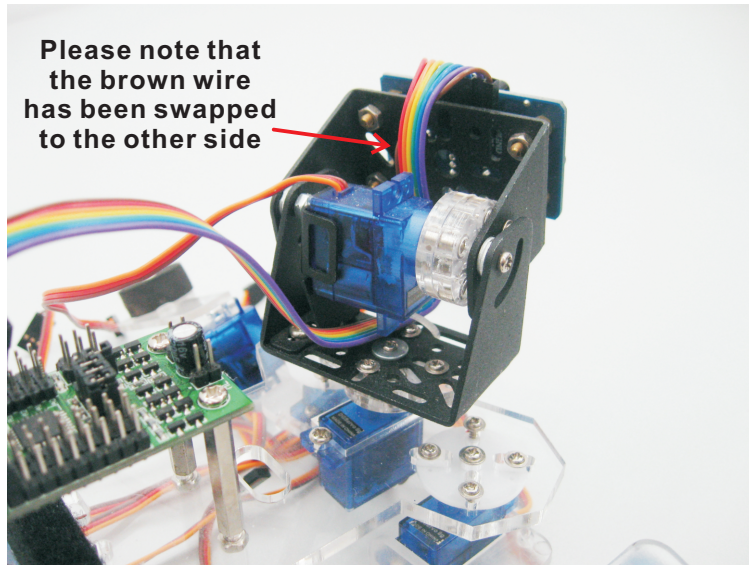
Step 13:



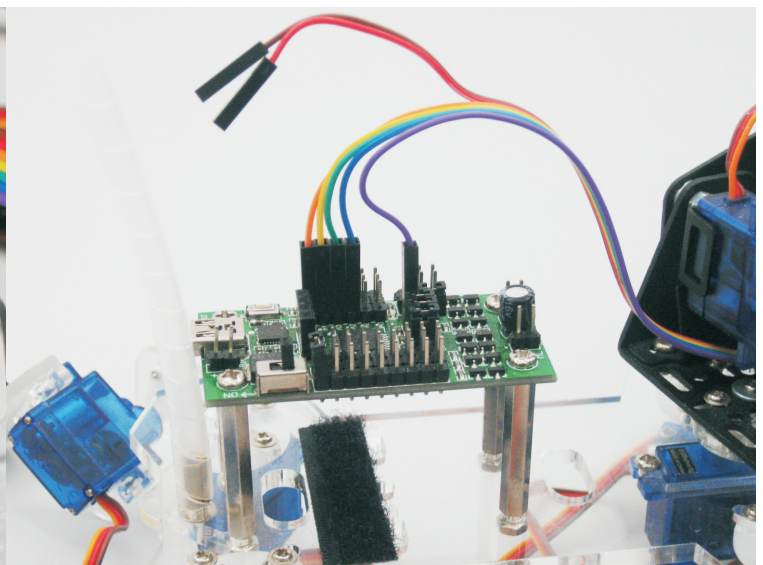
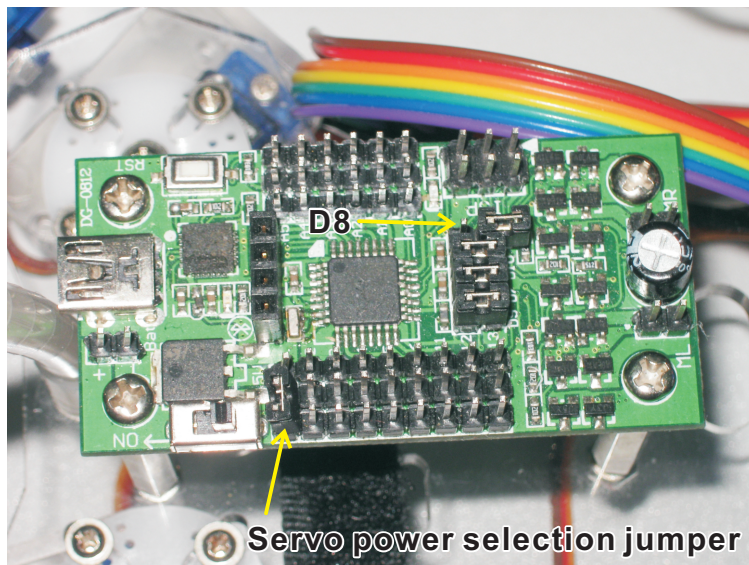
Ready for wiring!

Wiring the robot:

Start by plugging the rainbow cable into the eye. Pay careful attention to the colours to ensure correct function of the eye. Thread the cable under the tilt servo and use a cable tie to hold it in place. Leave this cable tie very loose so that the cable can move a bit if necessary.



The Mini Driver includes a dual motor driver that is not used for this robot. By disconnecting the jumpers we can access the control pins. Disconnect the Jumper for D8 and connect the violet wire. This wire controls the IR LEDs on the compound eye. Now connect the blue, green, yellow and orange wires to A2, A3, A4 and A5 as shown. These are your analog signals from the compound eye.

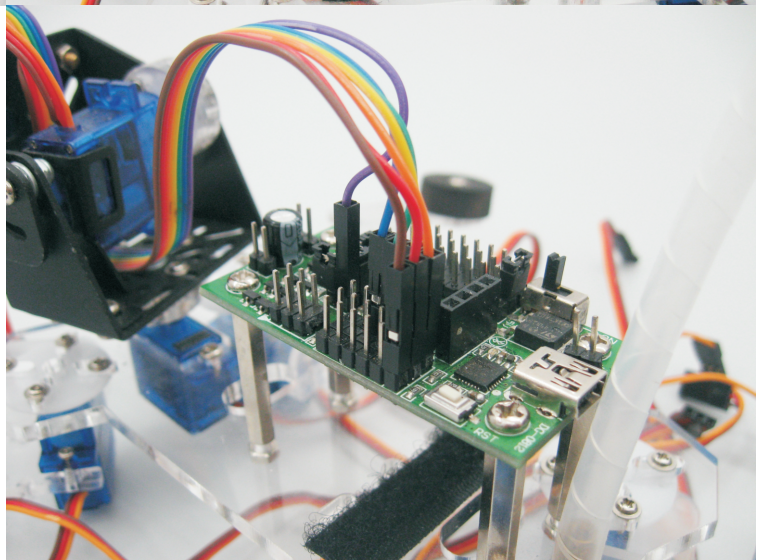


Make sure the servo power selection jumper is toward the outer edge of the PCB. This will power the leg servos directly from the battery.

When servos are not being used this jumper can be set to 5V allowing digital sensors to be powered from the 5V regulator instead.

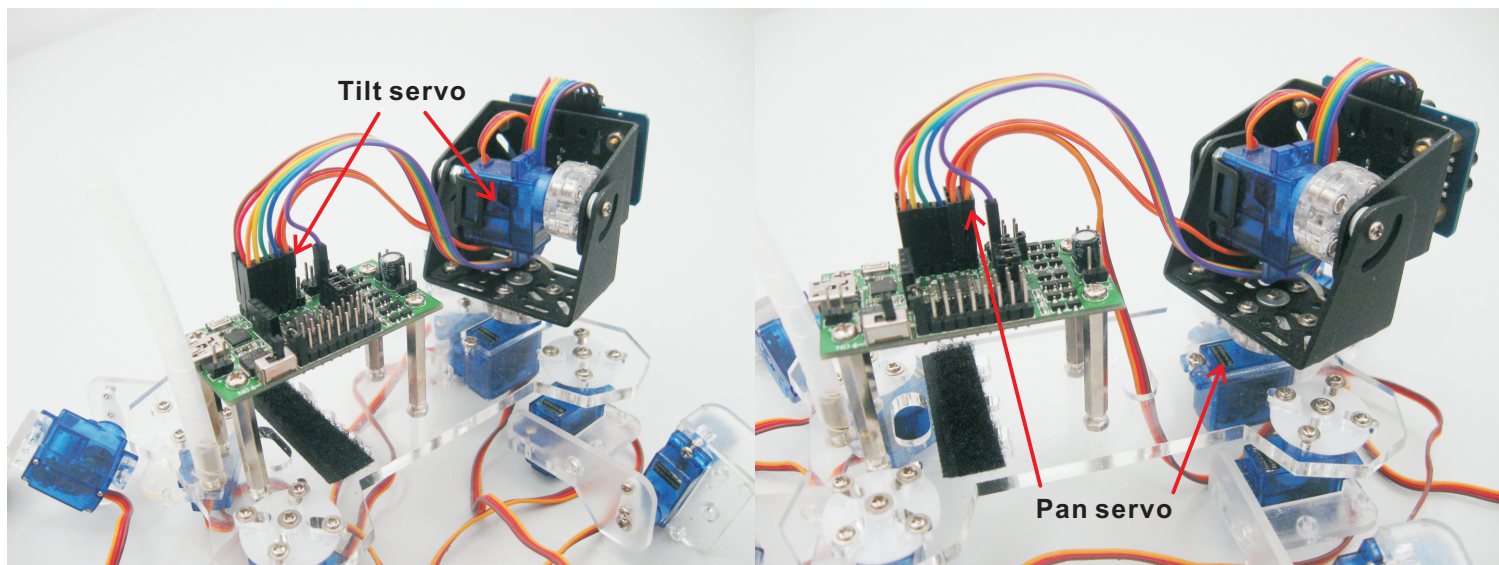
Now connect the power wires for the compound eye. Red is +5V and brown is Gnd. Check carefully that these wires are connected as shown in the photo with the brown wire closest to the edge of the PCB.

The power pins on the analog inputs are always connected to the 5V regulator no matter what position the servo power jumper is in.

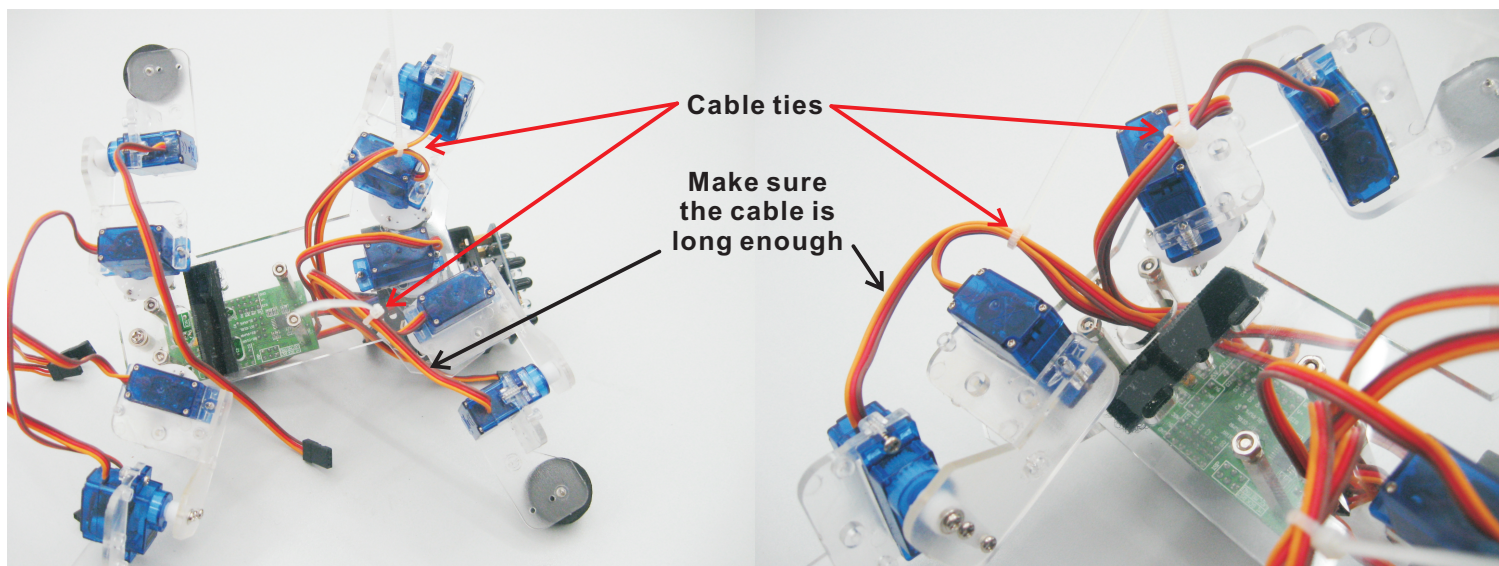


In this robot, A0 and A1 are being used as digital outputs instead of analog inputs to drive the pan and tilt servos. These servos are getting power through the 5V regulator so they cannot be heavily loaded.

Connect the "tilt" servo to A1 as shown in the first photo. Make sure the brown ground wire is closest to the outer edge of the PCB. Next connect the "pan" servo to A0 as shown in the second photo.

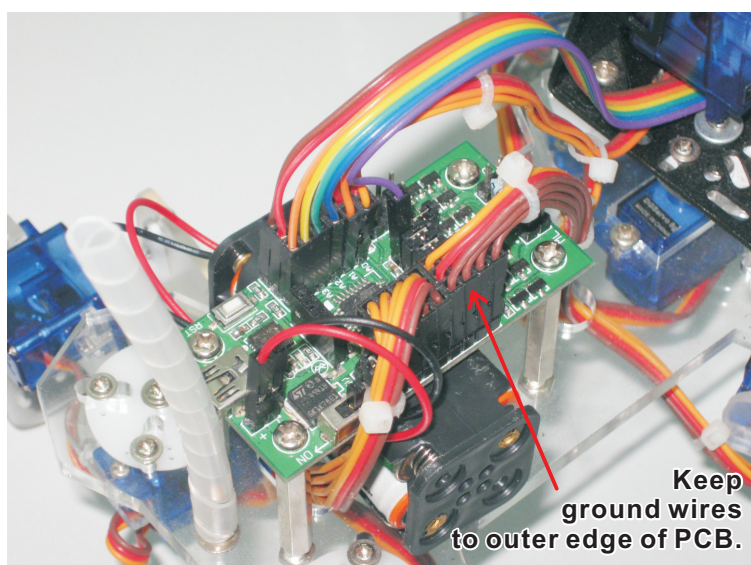


Loosely cable tie the front knee and hip servos as shown. Make sure the knee and hip can both move freely over their entire range before tightening the cable tie. Repeat the process for the rear legs. Feed these cables through the holes in the base plate as shown.



Connect the hip and knee servos as shown below. Try to keep the cables neat as shown.

Front Left Hip	D6
Front Left Knee	D12
Front Right Hip	D11
Front Right Knee	D13
Rear Left Hip	D5
Rear Left Knee	D2
Rear Right Hip	D4
Rear Right Knee	D3



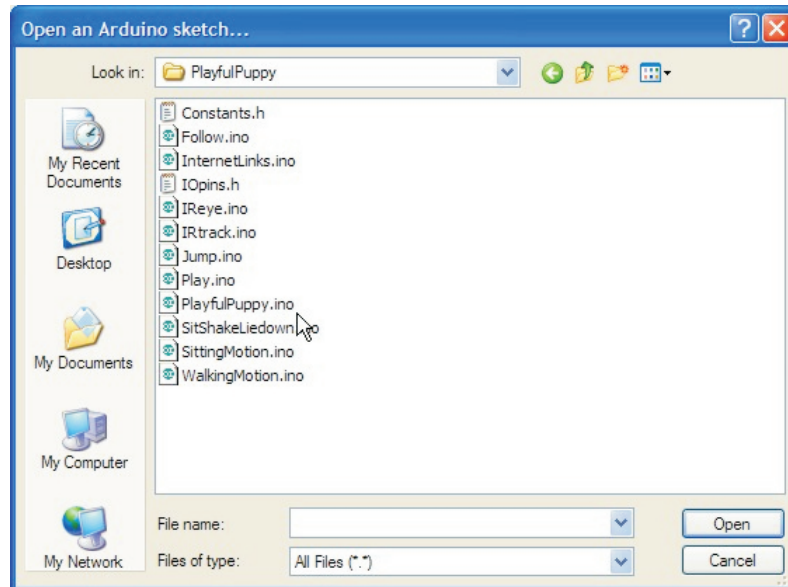
Installing the software:

The CD includes a copy of all third party software required for those without internet access. It is recommended that you check the internet for the latest versions. To install the sample code you must first have the Arduino IDE version 1.03 or later running on your computer. The Arduino IDE can be downloaded for free from here: <http://arduino.cc/en/Main/Software>. The Arduino IDE is available for Windows, Mac and Linux platforms.

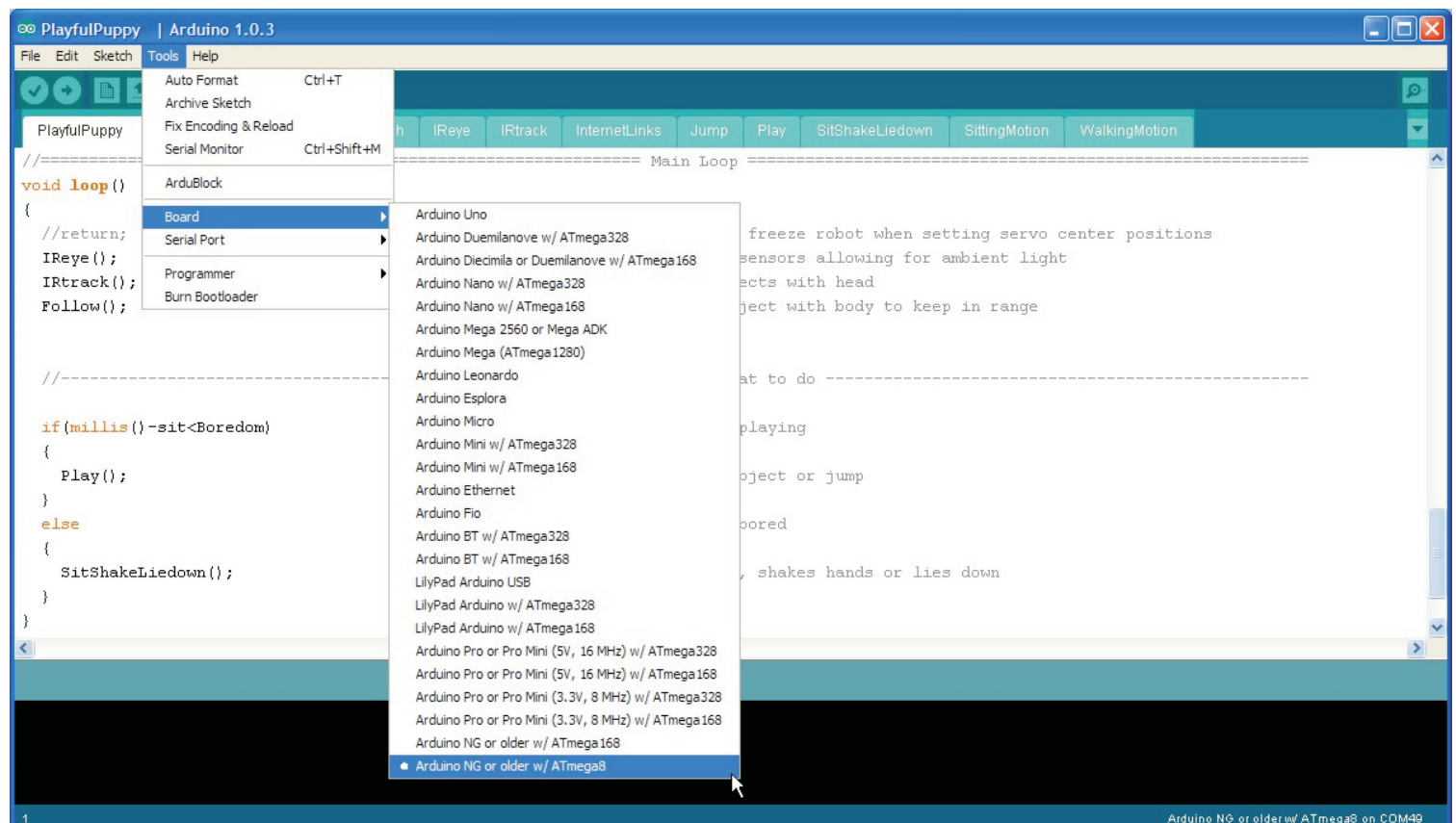
The robot controller uses the CP2102 USB interface IC. Depending on your OS you may need to install drivers. These are included on the CD or you can download the latest from here:

<http://www.silabs.com/products/mcu/Pages/USBtoUARTBridgeVCPDrivers.aspx>.

Once you have the Arduino IDE running you can goto the PlayfulPuppy folder and open the "PlayfulPuppy.ino" file on your CD. If your CD is lost or damaged you can download the software from the DAGU product support site here: <https://sites.google.com/site/daguproducts/>



The Playful Puppy robot uses the Mini Driver robot controller from DAGU. Before we can upload the sample code we must select the correct board type and serial port. This is done in the Tools menu. Select "Arduino NG or older w/ATmega8" as your board type.

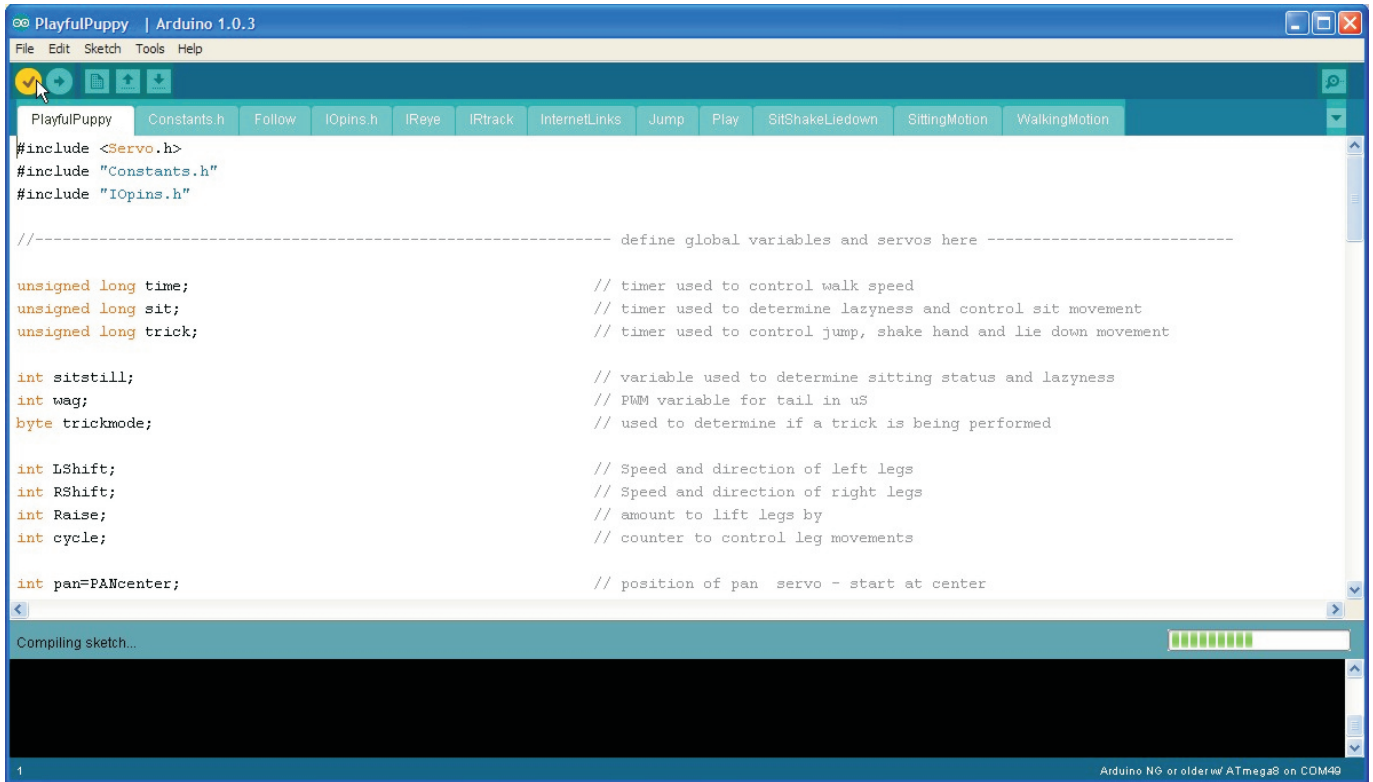


Installing the software (cont.):

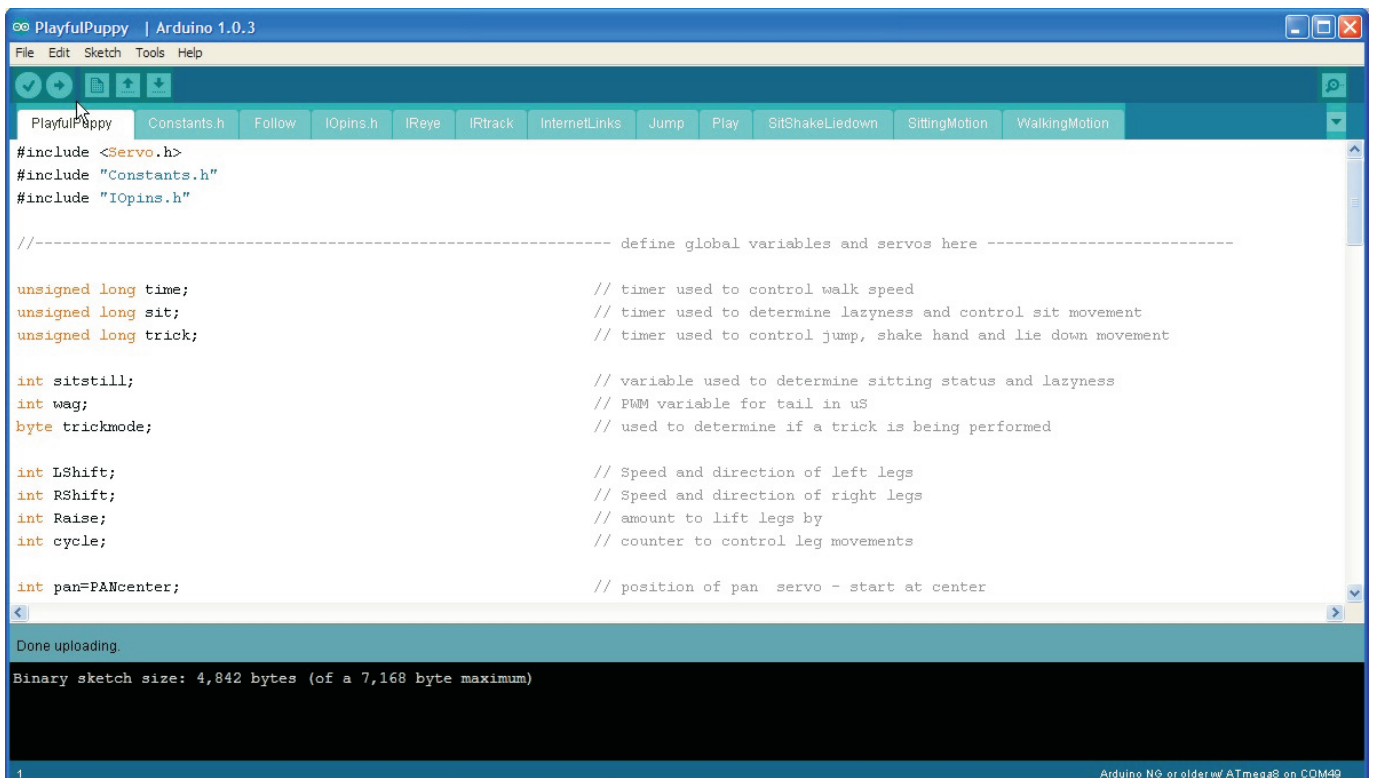
Before you plug the robot into the computer make sure you have 4x freshly charged AAA NiMh batteries installed and the power connector plugged in the correct way. Turn off the robot while uploading the new program to stop the robot from trying to move. The processor and head servos will be powered from the USB port but the leg servos will not have power.

If you are using a small laptop or your USB ports are heavily loaded then your computer may not be able to power the robot through the USB. In this case, leave the robot turned on and put it on it's back so it cannot try to run away.

Before you upload the code it is always good practice to verify it first. This will compile the code and check for any errors before attempting to upload the code. When you click on the upload button a progress bar will appear at the bottom. The message "Done compiling." and the size of the compiled code will appear when the process is complete.



Once the code has been verified you can upload the code to the robot. The progress bar will appear again and the D13 LED on the controller will flash a few times. Once the message "Done uploading." appears you can disconnect the robot and turn it on. The robot will not start immediately. This gives you time to put it on the ground.





Controlling the robot:

Using the sample code provided, the puppy responds to movement. When your hand or any object is in range (within 15cm - 20cm) of the puppy's compound eye the puppy will track the movement of your hand with it's head.

When standing the puppy will follow your hand and if necessary, will turn to try and keep your hand in front of it. If your hand gets too close the robot will back up. If your hand is too far away (but still in range) then the robot will walk forward.

If you do not play with the puppy it will get bored and sit down. While the robot is sitting, if you slowly move your hand down and then towards one of the front legs it will lift that leg to "Shake Hands". If you move your hand between the legs the robot will lie down.

To get the robot back up on it's feet you need to make it look straight up. Making the puppy look straight up will get it to jump up on it's back legs.



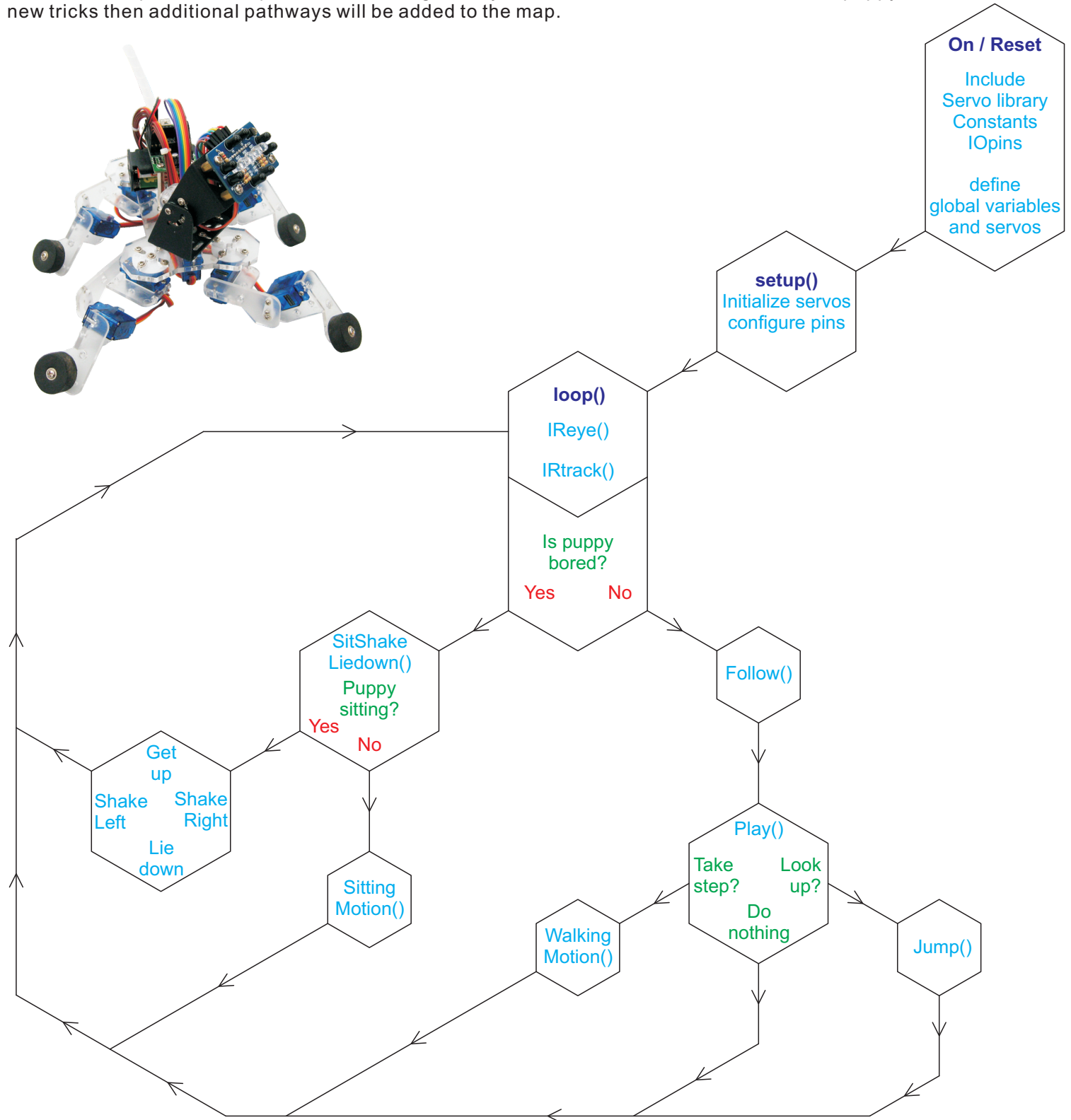
Understanding the Code:

Although you can play with the robot using just the sample code provided, the real fun is in teaching it new tricks! This section explains some of the basic features of the code and how they can be changed. Do not be afraid to experiment, you can always restore the original code if you make a mistake.

Below is a flowchart of the sample code. A flow chart is not actual code, it is more like a map showing blocks of code and the different paths the processor can follow within your code. Lines linking the blocks have an arrow head showing the direction that the processor must follow along the path.

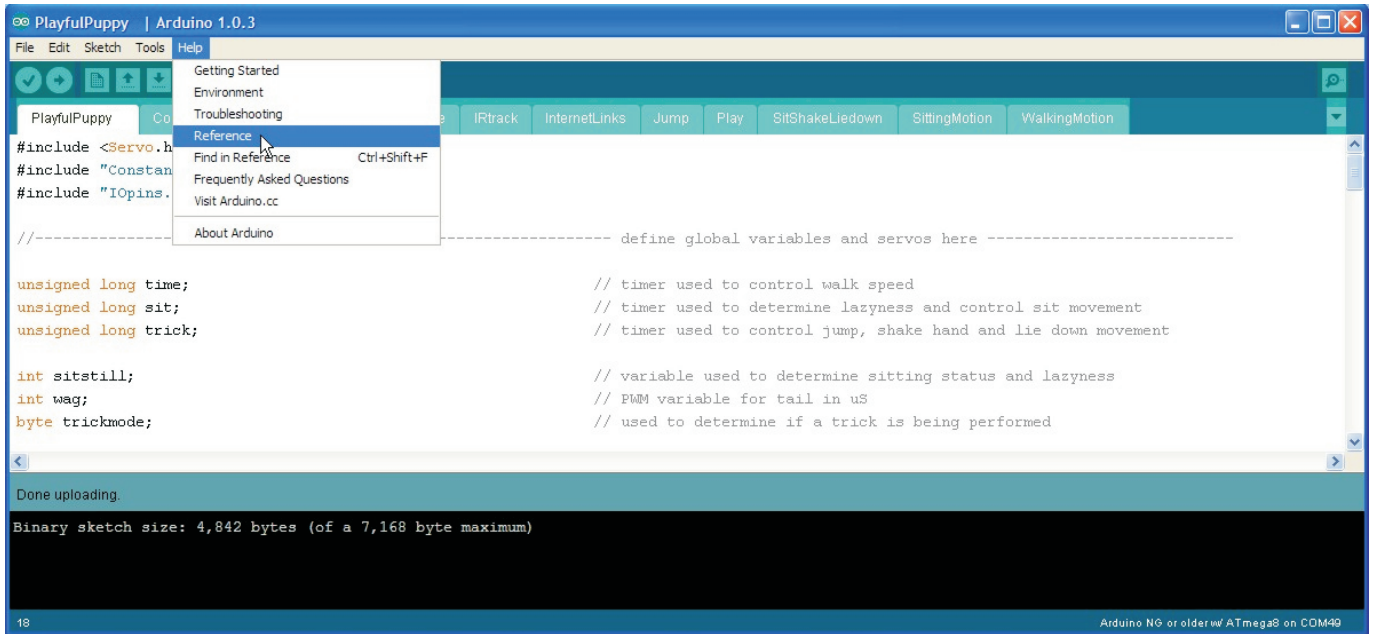
As you can see, once the processor enters the loop(), no matter which path it takes, they all return back to the loop() and the process is repeated. The only way to break out of the loop is to switch the power off or reset the processor.

Depending on the inputs from the sensors and the values of the timers, different paths are taken as the code repeats the loop over and over again. If you write additional code to teach the puppy new tricks then additional pathways will be added to the map.



Understanding the code (cont.)

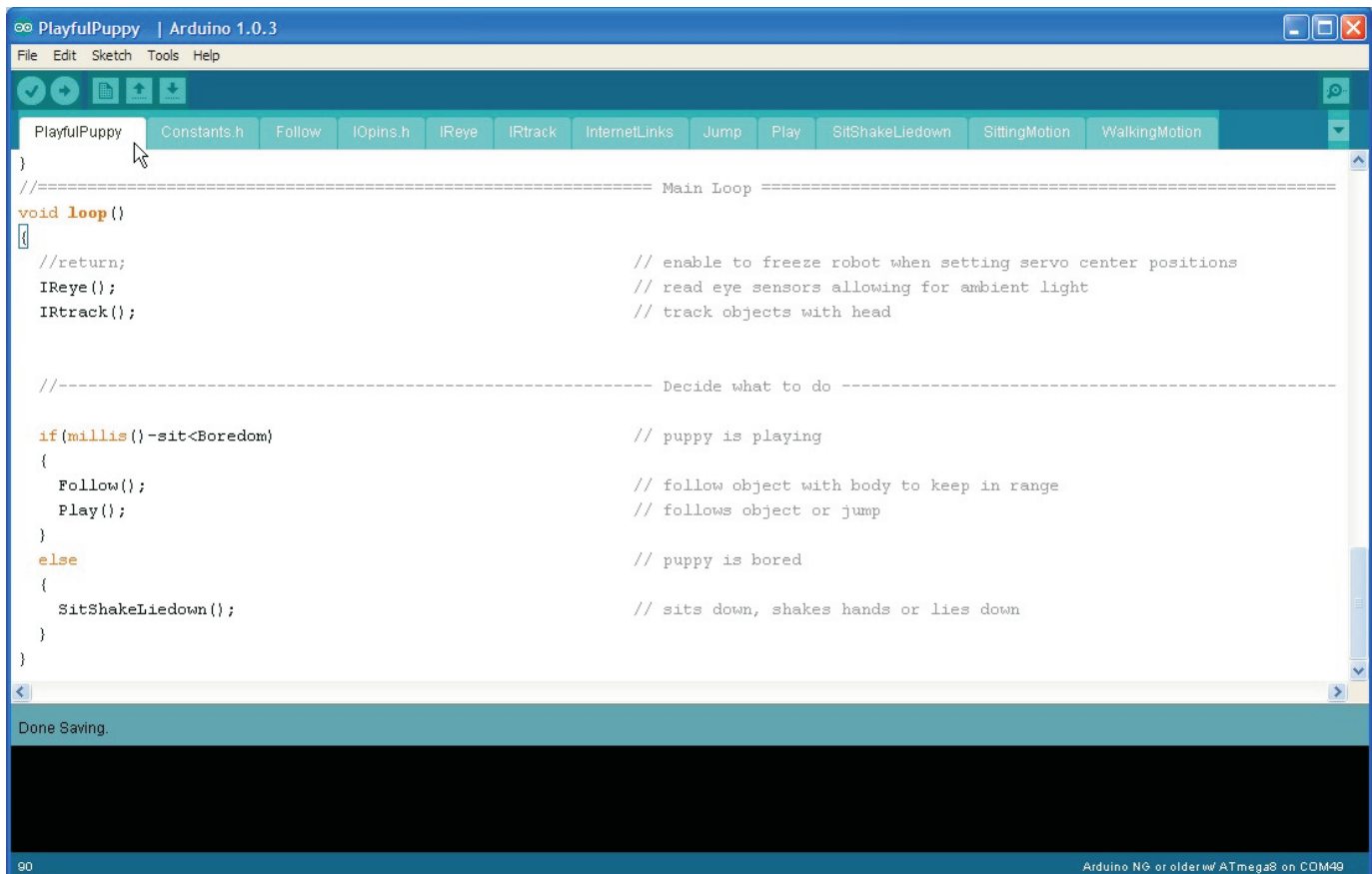
If you are new to Arduino then a guide to all the main commands and functions can be found by going to help and selecting reference.



The first tab is the main program. In the beginning the **<Servo.h>** library and definitions are added. Then global variables and servos are defined.

The **setup()** function runs only once when the program first runs. This is where the servos are attached to their digital pins and initialized to their center positions. By default, all digital pins are configured as inputs on startup as a safety precaution. The **pinMode()** command is used here to reconfigure the **IRleds** pin to become a digital output so it can control the IR LEDs on the compound eye.

The **loop()** function is the main section of code. As the name suggest, this code is run repeatedly until the processor is reset or turned off.

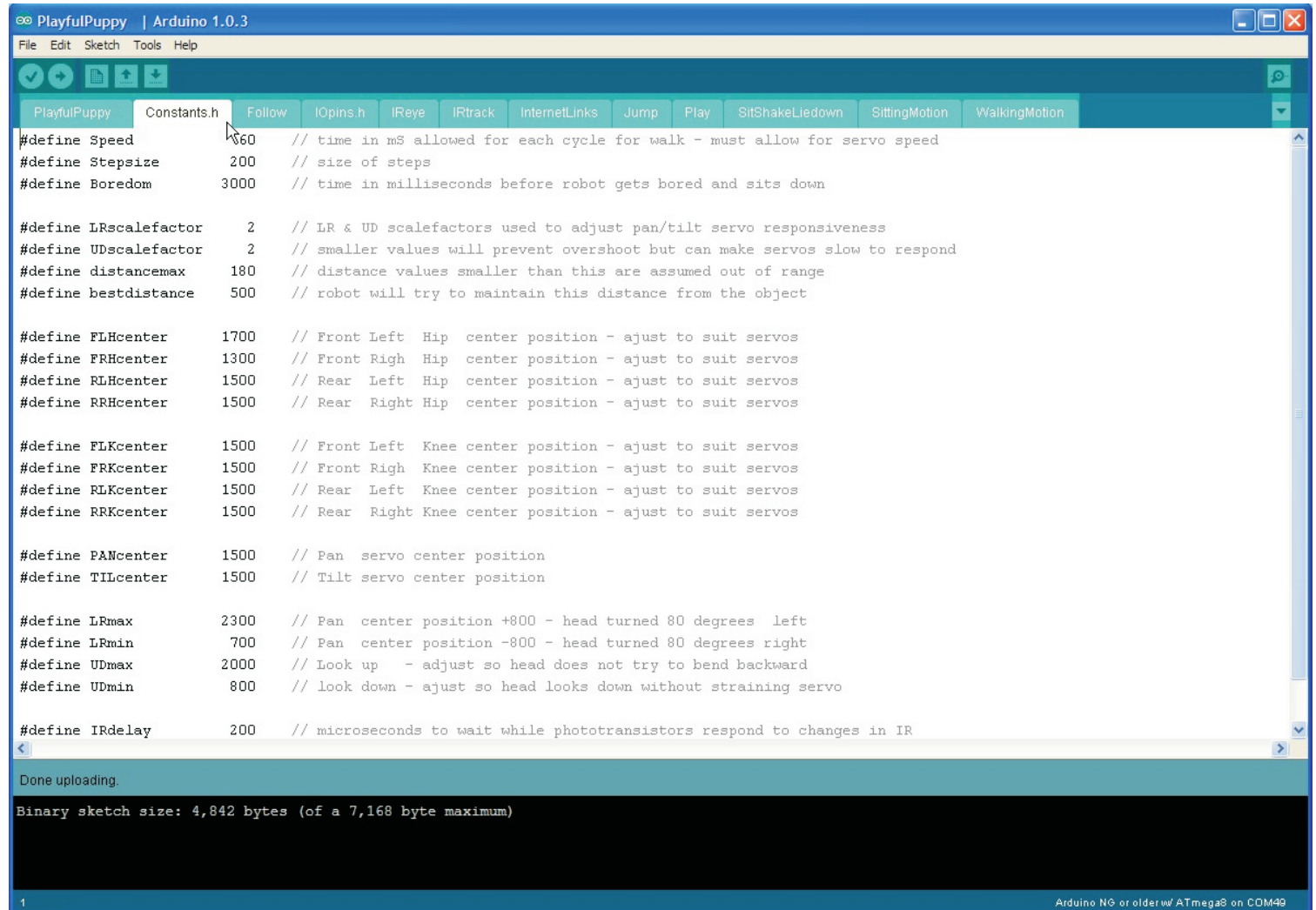


Understanding the code (cont.)

The second tab is **[Constants.h]**. This tab contains definitions for constant values that do not change when the program is running. In this tab we can set the center position of each servo and experiment with some other values that affect the way the robot walks and tracks movement.

Normally the servos should be centered at 1500. If they need a big adjustment you should re-seat the servo horn. For fine adjustment (+/- 200) you can change the center values in this tab.

The **Boredom** constant determines how long the robot will wait for you to play with it before it gets bored and sits down. 3000 milliseconds = 3 seconds.



```
PlayfulPuppy | Arduino 1.0.3
File Edit Sketch Tools Help

PlayfulPuppy Constants.h Follow IOPins.h IReye IRtrack InternetLinks Jump Play SitShakeLiedown SittingMotion WalkingMotion

#define Speed 60 // time in mS allowed for each cycle for walk - must allow for servo speed
#define Stepsize 200 // size of steps
#define Boredom 3000 // time in milliseconds before robot gets bored and sits down

#define LRscalefactor 2 // LR & UD scalefactors used to adjust pan/tilt servo responsiveness
#define UDscalefactor 2 // smaller values will prevent overshoot but can make servos slow to respond
#define distancemax 180 // distance values smaller than this are assumed out of range
#define bestdistance 500 // robot will try to maintain this distance from the object

#define FLHcenter 1700 // Front Left Hip center position - ajust to suit servos
#define FRHcenter 1300 // Front Righ Hip center position - ajust to suit servos
#define RLHcenter 1500 // Rear Left Hip center position - ajust to suit servos
#define RRHcenter 1500 // Rear Right Hip center position - ajust to suit servos

#define FLKcenter 1500 // Front Left Knee center position - ajust to suit servos
#define FRKcenter 1500 // Front Righ Knee center position - ajust to suit servos
#define RLKcenter 1500 // Rear Left Knee center position - ajust to suit servos
#define RRKcenter 1500 // Rear Right Knee center position - ajust to suit servos

#define PANcenter 1500 // Pan servo center position
#define TILcenter 1500 // Tilt servo center position

#define LRmax 2300 // Pan center position +800 - head turned 80 degrees left
#define LRmin 700 // Pan center position -800 - head turned 80 degrees right
#define UDmax 2000 // Look up - adjust so head does not try to bend backward
#define UDmin 800 // look down - ajust so head looks down without straining servo

#define IRdelay 200 // microseconds to wait while phototransistors respond to changes in IR

Done uploading.
Binary sketch size: 4,842 bytes (of a 7,168 byte maximum)

1 Arduino NG or older w/ ATmega8 on COM49
```

The **speed** and **stepsize** constants can be used to change the gait of the robot. Bigger steps need more time for the servo to move but cover more ground. Smaller steps can be completed quicker and might offer smoother movement. Making the speed value smaller will make the robot faster but if it is too small then the servos will not keep up and the robot will not walk properly.

The constant **bestdistance** is the average distance the robot tries to maintain between itself and the object it is tracking. Bigger values = smaller distance. Closer distances make it easier to maintain a lock on the object.

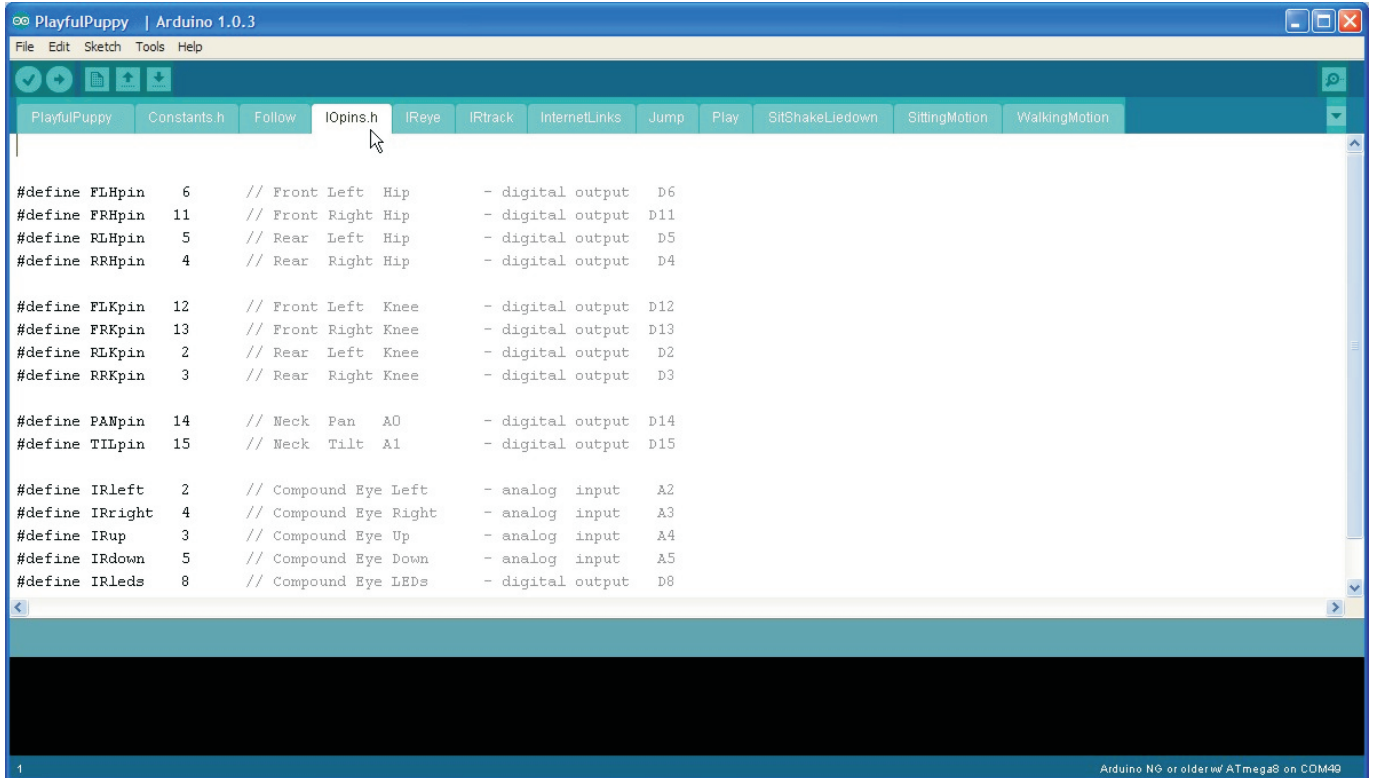
The constant **distancemax** is the maximum distance an object can be before the robot ignores it. If this value is too small then the robot will confuse ambient light for an object. The smaller the value, the greater the distance.

LRscalefactor and **UDscalefactor** are used to adjust how quickly the head moves in response to an object. In most cases you do not need to adjust these values. If the head moves too quickly it can overshoot and the head will tend to vibrate or constantly shake. Too slow and the robot will loose it's lock on a moving object. If you make changes to the code that affect how quickly the main loop repeats then these values may need adjustment to compensate.

IRdelay is the time in microseconds needed for the photo transistors in the compound eye to respond to changes in the IR light when the IR LEDs turn on or off. Increasing this value may improve the robot's sensitivity a bit but it will also make the robot slower and less responsive.

Understanding the code (cont.)

The [IOpins.h] tab is your programs wiring diagram. If you want to change which pins are used for a servo or sensor then that change must be reflected here. If you need to replace a servo or even if you are wiring up the robot for the first time then this tab is your guide as to what device connects to which pin.



```
#define FLHpin 6 // Front Left Hip - digital output D6
#define FRHpin 11 // Front Right Hip - digital output D11
#define RLHpin 5 // Rear Left Hip - digital output D5
#define RRHpin 4 // Rear Right Hip - digital output D4

#define FLKpin 12 // Front Left Knee - digital output D12
#define FRKpin 13 // Front Right Knee - digital output D13
#define RLKpin 2 // Rear Left Knee - digital output D2
#define RRKpin 3 // Rear Right Knee - digital output D3

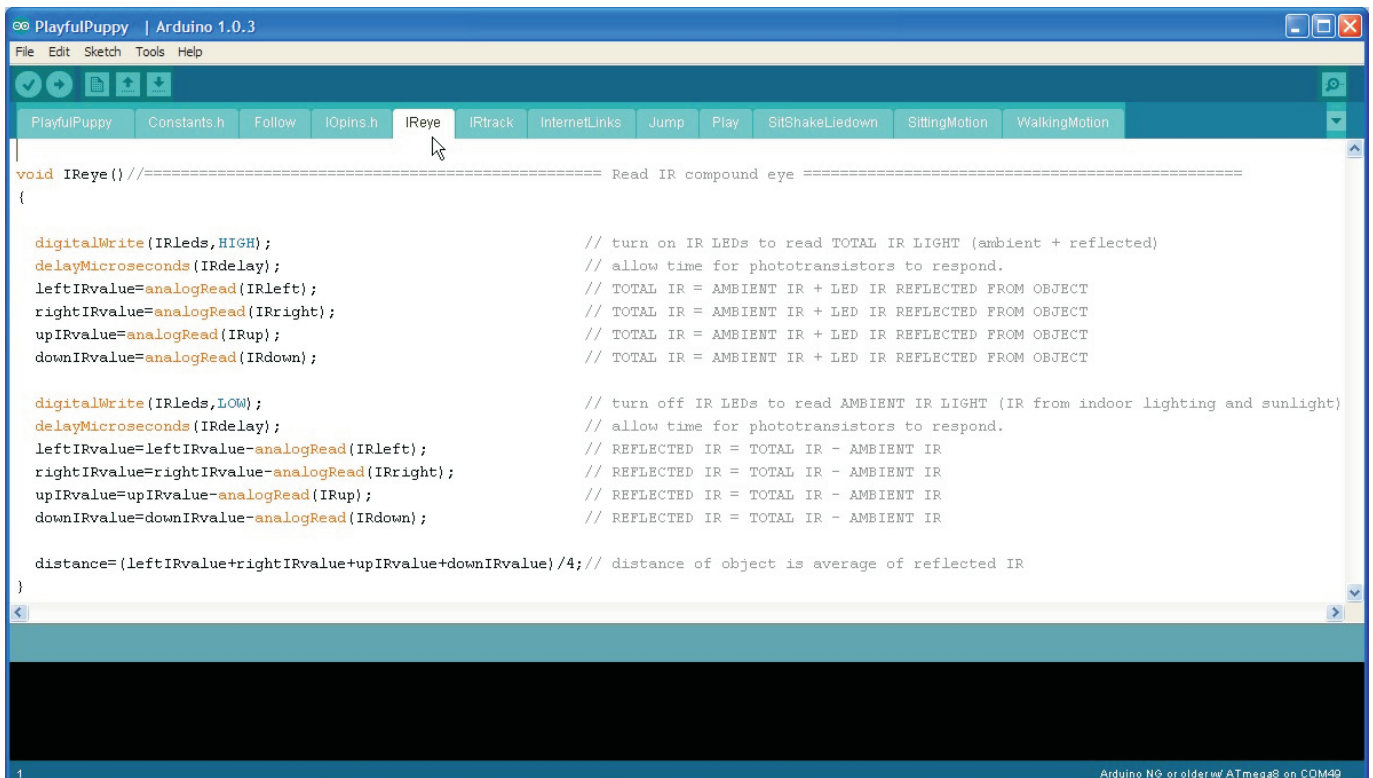
#define PANpin 14 // Neck Pan A0 - digital output D14
#define TILpin 15 // Neck Tilt A1 - digital output D15

#define IRLeft 2 // Compound Eye Left - analog input A2
#define IRight 4 // Compound Eye Right - analog input A3
#define IRup 3 // Compound Eye Up - analog input A4
#define IRdown 5 // Compound Eye Down - analog input A5
#define IRLeds 8 // Compound Eye LEDs - digital output D8
```

The [IReye] tab is where the data from the eye is generated. This code first takes a reading of all sensors with the IR LEDs turned ON. This reading is a combination of the ambient IR light plus the IR light from the LEDs being reflected from any nearby objects.

Then the LEDs are turned OFF and the sensors are read again. This reading is only the ambient IR light. These values are subtracted from the previous sensor readings to give a value equal to just the IR light from the IR LEDs being reflected by any nearby objects.

This value depends a lot on the size and colour of the object being tracked. If there is a lot of ambient IR light such as bright sunlight then the sensor can be blinded. The eye works best indoors or at night.



```
void IReye{//===== Read IR compound eye =====
{
    digitalWrite(IRLeds,HIGH); // turn on IR LEDs to read TOTAL IR LIGHT (ambient + reflected)
    delayMicroseconds(IRdelay); // allow time for phototransistors to respond.
    leftIRvalue=analogRead(IRLeft); // TOTAL IR = AMBIENT IR + LED IR REFLECTED FROM OBJECT
    rightIRvalue=analogRead(IRright); // TOTAL IR = AMBIENT IR + LED IR REFLECTED FROM OBJECT
    upIRvalue=analogRead(IRup); // TOTAL IR = AMBIENT IR + LED IR REFLECTED FROM OBJECT
    downIRvalue=analogRead(IRdown); // TOTAL IR = AMBIENT IR + LED IR REFLECTED FROM OBJECT

    digitalWrite(IRLeds,LOW); // turn off IR LEDs to read AMBIENT IR LIGHT (IR from indoor lighting and sunlight)
    delayMicroseconds(IRdelay); // allow time for phototransistors to respond.
    leftIRvalue=leftIRvalue-analogRead(IRLeft); // REFLECTED IR = TOTAL IR - AMBIENT IR
    rightIRvalue=rightIRvalue-analogRead(IRright); // REFLECTED IR = TOTAL IR - AMBIENT IR
    upIRvalue=upIRvalue-analogRead(IRup); // REFLECTED IR = TOTAL IR - AMBIENT IR
    downIRvalue=downIRvalue-analogRead(IRdown); // REFLECTED IR = TOTAL IR - AMBIENT IR

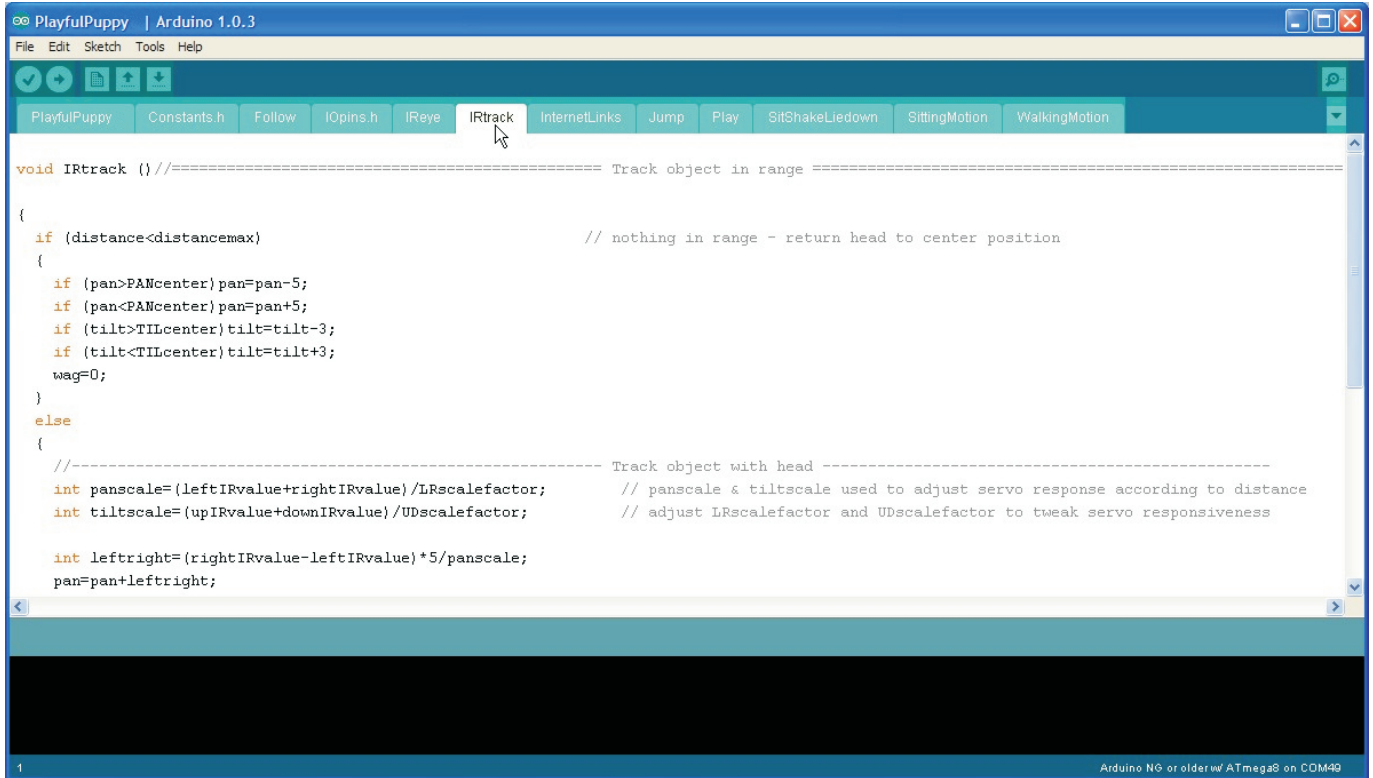
    distance=(leftIRvalue+rightIRvalue+upIRvalue+downIRvalue)/4;// distance of object is average of reflected IR
}
```


Understanding the code (cont.)

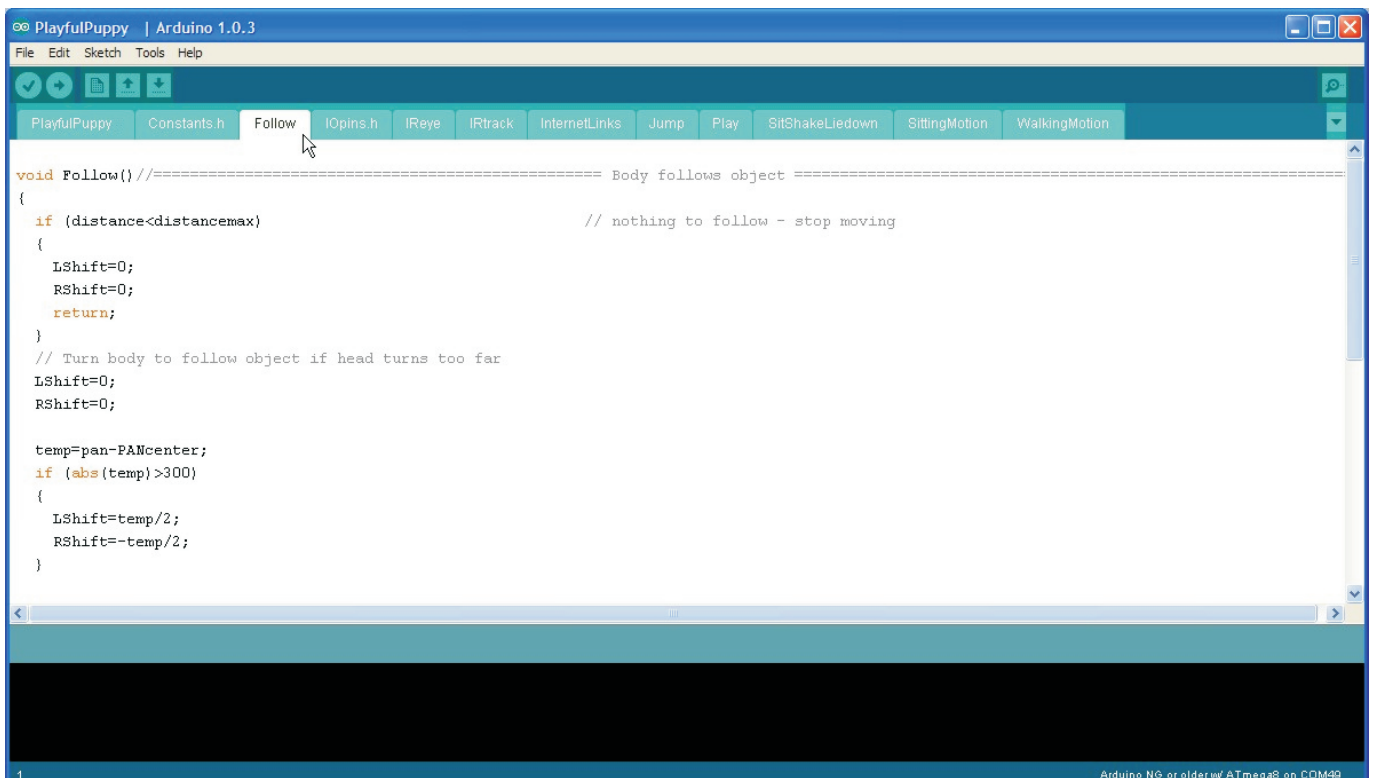
The tab **[IRtrack]** is where the data from the compound eye is used to control the pan and tilt servos. If there are no objects nearby then the head will slowly return to center position.

Basically the code adjust the pan servo to keep the left and right IR values equal. The tilt servo is adjusted to keep the upper and lower IR values equal. This keeps the sensor facing directly at any nearby object.

The variables **panscale** and **tiltscale** are used to try and compensate for distance and the speed at which the program runs. The constants, **LRscalefactor** and **UDscalefactor** can be changed in the **[Constants.h]** tab to compensate for changes in the code that affect the speed of the code.

A screenshot of the Arduino IDE interface. The top menu bar shows 'File', 'Edit', 'Sketch', 'Tools', and 'Help'. Below the menu bar is a toolbar with icons for saving, opening, and running. The tab bar at the top lists several tabs: 'PlayfulPuppy', 'Constants.h', 'Follow', 'IOpins.h', 'iReye', 'IRtrack' (which is selected), 'InternetLinks', 'Jump', 'Play', 'SitShakeLiedown', 'SittingMotion', and 'WalkingMotion'. The main code area displays the 'IRtrack' function. The code starts with a comment 'Track object in range' and an empty curly brace. It then has an 'if' statement checking 'distance < distancemax'. Inside this 'if' block, there are four 'if' statements for adjusting 'pan' and 'tilt' values based on comparisons with 'PANcenter' and 'TILcenter', followed by a 'wag=0;' statement. An 'else' block follows, containing a comment 'Track object with head' and two 'int' declarations for 'panscale' and 'tiltscale' using 'LRscalefactor' and 'UDscalefactor' respectively. It then calculates 'leftright' and updates 'pan'. The status bar at the bottom indicates '1' and 'Arduino NG or older w/ ATmega8 on COM49'.

The code in the **[Follow]** tab is where the speed and direction of the legs are controlled. If the head has to turn too far to track an object, the **LShift** and **RShift** walking speeds are adjusted to turn the body towards the object being tracked. These speeds are then adjusted forward or backward to try and maintain **bestdistance** from the object being tracked.

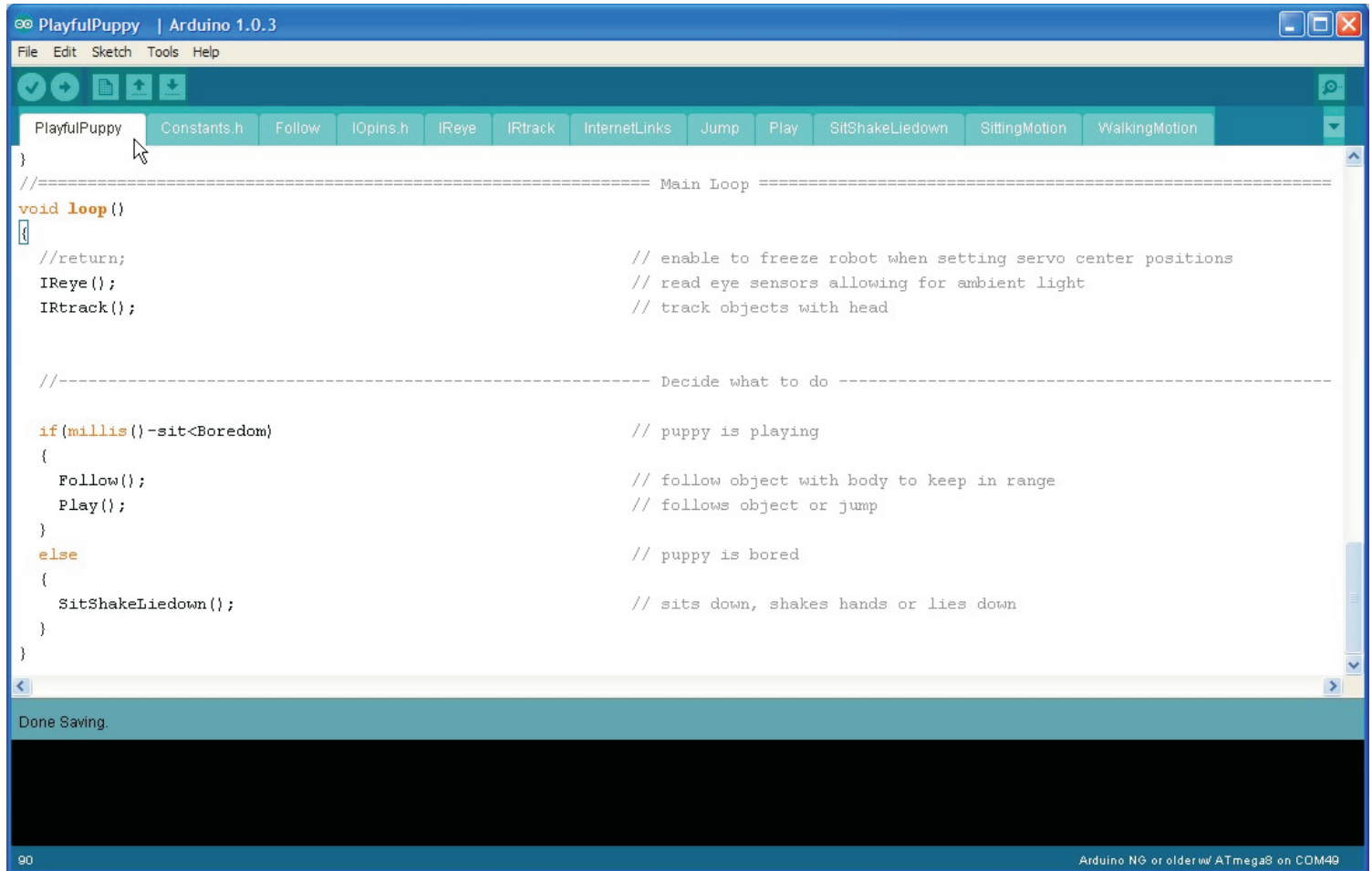
A screenshot of the Arduino IDE interface, similar to the previous one, but with the 'Follow' tab selected. The 'Follow' function is shown in the code area. It starts with a comment 'Body follows object' and an empty curly brace. An 'if' statement checks 'distance < distancemax'. Inside, 'LShift' and 'RShift' are set to 0, and 'return;' is called. A comment '// Turn body to follow object if head turns too far' is followed by 'LShift=0;' and 'RShift=0;'. Then, 'temp' is assigned 'pan-PANcenter', and an 'if' statement checks 'abs(temp) > 300'. Inside this 'if' block, 'LShift' is set to 'temp/2' and 'RShift' is set to '-temp/2'. The status bar at the bottom shows '1' and 'Arduino NG or older w/ ATmega8 on COM49'.

Understanding the code (cont.)

This robot uses some timers in the code to control certain functions. This is done using the `millis()` function which counts the number of milliseconds since the robot was turned on. We will briefly look at how the unsigned long variable **sit** is used to measure boredom in the robot.

In the **[Play]** tab, **sit** is set to equal the value reported by the **millis()** function whenever the robot takes a step. This resets the boredom timer because the difference between **millis()** and **sit** is reset to 0.

In the **loop()** section of the code, the difference between **millis()** and **sit** are compared to the constant **Boredom**. The result is that if the robot does not take a step for **Boredom** milliseconds then the robot will become bored and sit down. Once this happens, the timer cannot be reset unless you make the robot look up at which point the variable **sit** is set to equal the new value of **millis()** in the **[SitShakeLiedown]** part of the code and the difference is now 0 again.



```
PlayfulPuppy | Arduino 1.0.3
File Edit Sketch Tools Help

PlayfulPuppy Constants.h Follow IOpins.h IReye IRtrack InternetLinks Jump Play SitShakeLiedown SittingMotion WalkingMotion

}

//===== Main Loop =====

void loop()
{
  //return; // enable to freeze robot when setting servo center positions
  IReye(); // read eye sensors allowing for ambient light
  IRtrack(); // track objects with head

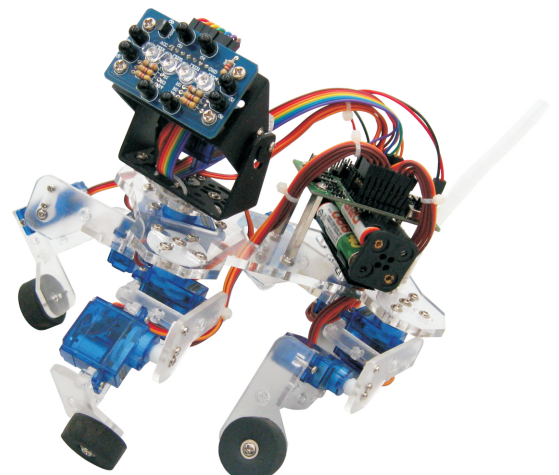
  //----- Decide what to do -----

  if(millis()-sit<Boredom) // puppy is playing
  {
    Follow(); // follow object with body to keep in range
    Play(); // follows object or jump
  }
  else // puppy is bored
  {
    SitShakeLiedown(); // sits down, shakes hands or lies down
  }
}
```

Done Saving.

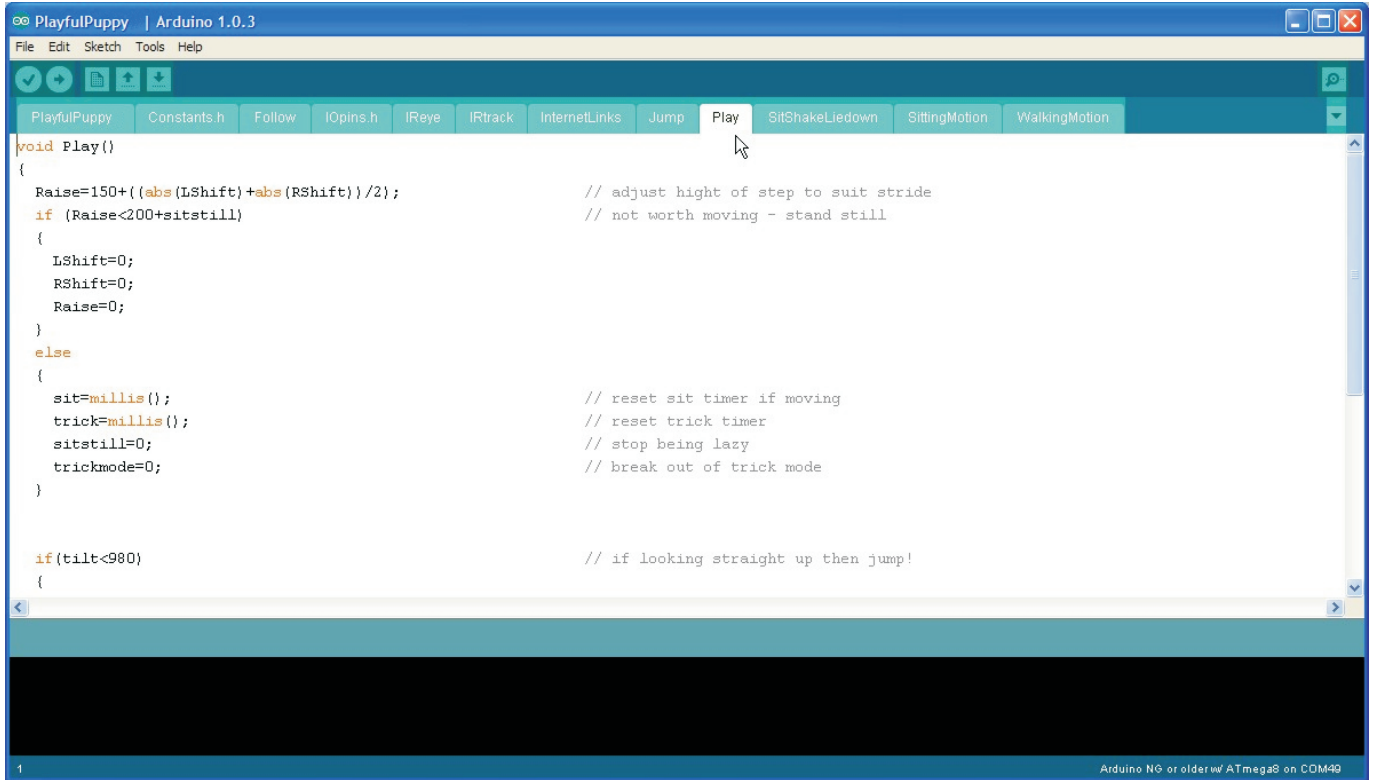
90 Arduino NG or older w/ ATmega8 on COM49

In the `loop()` function, the program constantly reads the sensors in the eye, tracks movement with the head. If the puppy is not bored then it will follow your hand and walk / jump in play mode. If the puppy becomes bored then it will sit down. In this mode it will shake hands or lie down.



Understanding the software (cont.)

If the robot is not bored then the code in the **[Play]** tab determines if the puppy walks or jumps. Normally the puppy will walk unless the size of the steps, determined by the distance is too small. If the head tilts far enough upward to track the object then the robot will jump.

A screenshot of the Arduino IDE interface. The title bar reads "PlayfulPuppy | Arduino 1.0.3". The menu bar includes "File", "Edit", "Sketch", "Tools", and "Help". Below the menu bar is a toolbar with icons for saving, running, and other functions. A series of tabs are visible: "PlayfulPuppy", "Constants.h", "Follow", "IOpins.h", "IReye", "IRtrack", "InternetLinks", "Jump", "Play", "SitShakeLiedown", "SittingMotion", and "WalkingMotion". The "Play" tab is currently selected. The code in this tab is as follows:

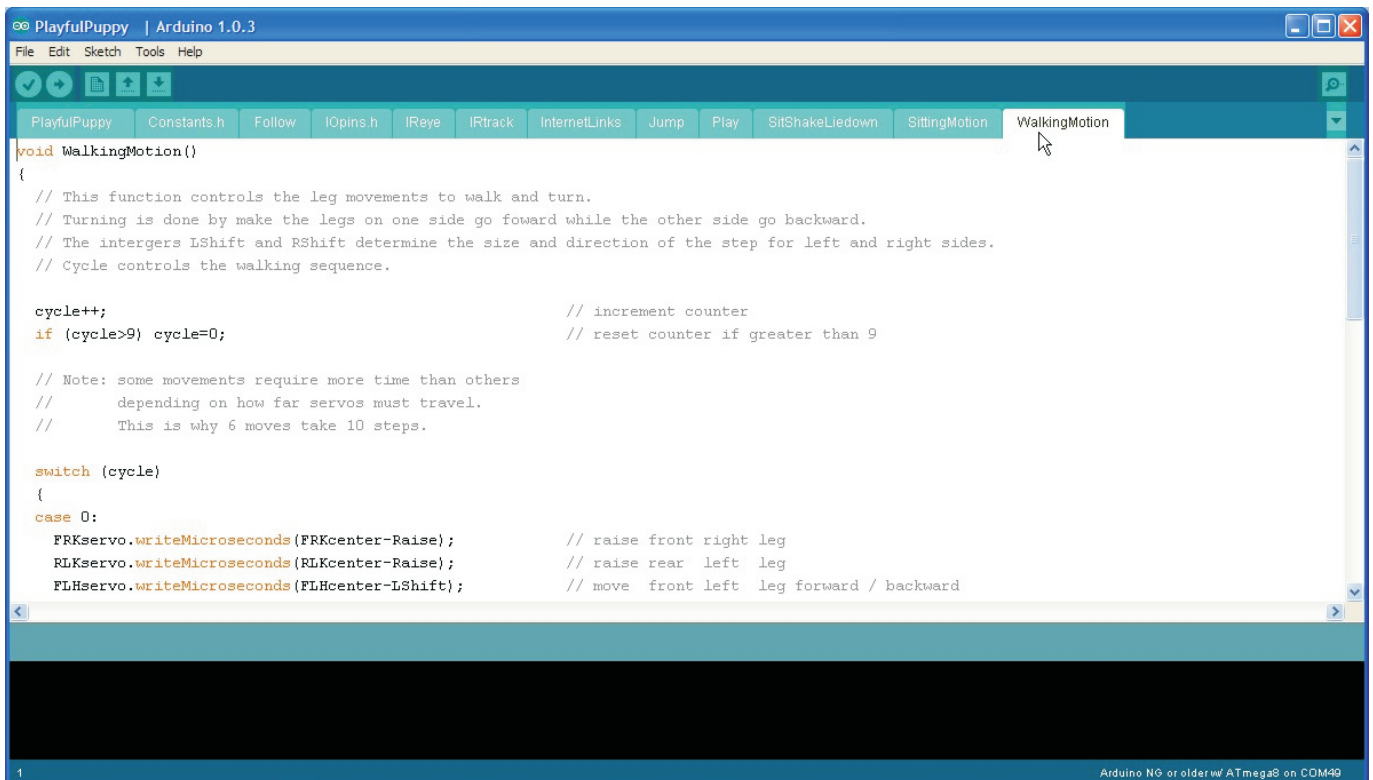
```
void Play()
{
  Raise=150+((abs(LShift)+abs(RShift))/2);          // adjust hight of step to suit stride
  if (Raise<200+sitstill)                          // not worth moving - stand still
  {
    LShift=0;
    RShift=0;
    Raise=0;
  }
  else
  {
    sit=millis();                                  // reset sit timer if moving
    trick=millis();                                // reset trick timer
    sitstill=0;                                    // stop being lazy
    trickmode=0;                                   // break out of trick mode
  }

  if(tilt<980)                                     // if looking straight up then jump!
  {
```

The bottom status bar indicates "Arduino NG or older w/ ATmega8 on COM49".

The code for the puppy's walking motion (gait) is in the **[WalkingMotion]** tab. The gait consist of 10 steps with nothing happening in steps 2,4,7 & 9 to allow more time for the legs to move forward / backward. Essentially, two diagonally opposite legs lift simultaneously, move forward or backward by the values **LShift**, **RShift** then lower again. Then the other two legs raise, move and lower.

Negative values of **LShift** and **RShift** make the legs go backward instead of forward. The **Speed** constant determines the number of milliseconds between steps in the gait. **Stepsize** set the limit for the size of the steps.

A screenshot of the Arduino IDE interface, similar to the previous one, but with the "WalkingMotion" tab selected. The code in this tab is as follows:

```
void WalkingMotion()
{
  // This function controls the leg movements to walk and turn.
  // Turning is done by make the legs on one side go foward while the other side go backward.
  // The intergers LShift and RShift determine the size and direction of the step for left and right sides.
  // Cycle controls the walking sequence.

  cycle++;                                          // increment counter
  if (cycle>9) cycle=0;                           // reset counter if greater than 9

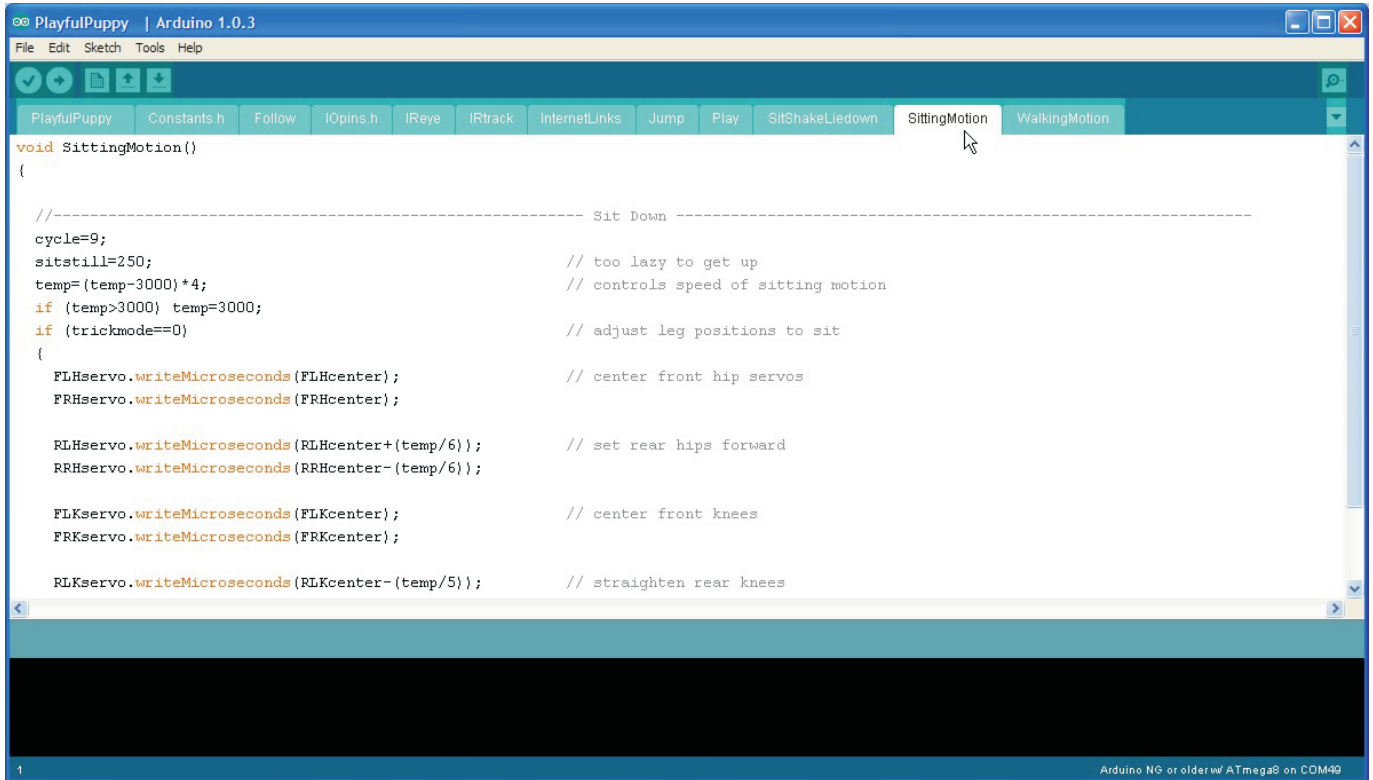
  // Note: some movements require more time than others
  // depending on how far servos must travel.
  // This is why 6 moves take 10 steps.

  switch (cycle)
  {
  case 0:
    FRKservo.writeMicroseconds(FRKcenter-Raise); // raise front right leg
    RLKservo.writeMicroseconds(RLKcenter-Raise); // raise rear left leg
    FLHservo.writeMicroseconds(FLHcenter-LShift); // move front left leg forward / backward
```

The bottom status bar indicates "Arduino NG or older w/ ATmega8 on COM49".

Understanding the software (cont.)

When the puppy first becomes bored, the code in the [**SittingMotion**] tab is called. So that the robot does not just suddenly drop down, the same timer used to measure boredom is used to control the robots sitting speed with the rear legs slowly straightening at the knee and moving forward at the hips.

A screenshot of the Arduino IDE interface. The title bar reads "PlayfulPuppy | Arduino 1.0.3". The menu bar includes "File", "Edit", "Sketch", "Tools", and "Help". Below the menu bar is a toolbar with icons for saving, running, and other functions. A tab bar at the top shows several tabs: "PlayfulPuppy", "Constants.h", "Follow", "IOpins.h", "IReye", "IRtrack", "InternetLinks", "Jump", "Play", "SitShakeLiedown", "SittingMotion" (which is currently selected), and "WalkingMotion". The main text area displays the code for the `void SittingMotion()` function. The code includes comments and servo control commands for the robot's legs and hips. The status bar at the bottom indicates "1" and "Arduino NG or older w/ ATmega8 on COM49".

```
void SittingMotion()
{
    //----- Sit Down -----
    cycle=9;
    sitstill=250;
    temp=(temp-3000)*4;
    if (temp>3000) temp=3000;
    if (trickmode==0)
    {
        FLHservo.writeMicroseconds(FLHcenter);
        FRHservo.writeMicroseconds(FRHcenter);

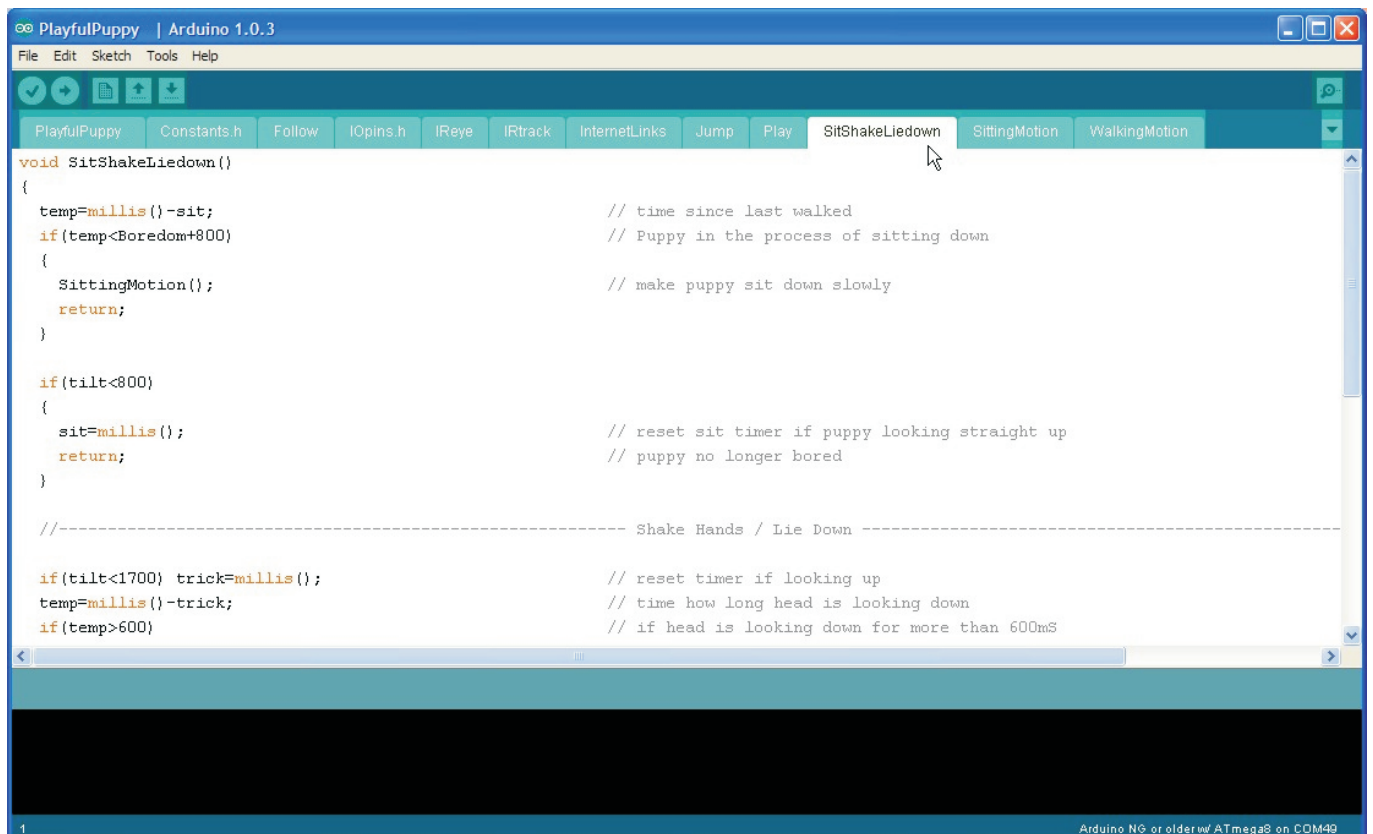
        RLHservo.writeMicroseconds(RLHcenter+(temp/6));
        RRHservo.writeMicroseconds(RRHcenter-(temp/6));

        FLKservo.writeMicroseconds(FLKcenter);
        FRKservo.writeMicroseconds(FRKcenter);

        RLKservo.writeMicroseconds(RLKcenter-(temp/5));
    }
}
```

Once the robot is sitting the code in the [**SitShakeLiedown**] tab determines the behavior of the puppy. While in the sitting position the robot will still track movement with it's head. If the head tracks movement towards either front leg then that leg will lift so that the puppy can shake hands. If the head tracks movement down and between the legs then both front legs will lift causing the puppy to lie down.

To break out of the sit, shake, lie down mode the robot must track movement upward causing it's head to tilt straight up. This will reset the boredom timer and put the puppy back into play mode.

A screenshot of the Arduino IDE interface, similar to the previous one. The title bar reads "PlayfulPuppy | Arduino 1.0.3". The menu bar is the same. The tab bar shows the same tabs, but "SitShakeLiedown" is now selected. The main text area displays the code for the `void SitShakeLiedown()` function. The code includes logic for handling the robot's state when it is sitting, including checks for boredom, head tilt, and head position relative to the legs. The status bar at the bottom indicates "1" and "Arduino NG or older w/ ATmega8 on COM49".

```
void SitShakeLiedown()
{
    temp=millis()-sit;
    if (temp<Boredom+800)
    {
        SittingMotion();
        return;
    }

    if (tilt<800)
    {
        sit=millis();
        return;
    }

    //----- Shake Hands / Lie Down -----

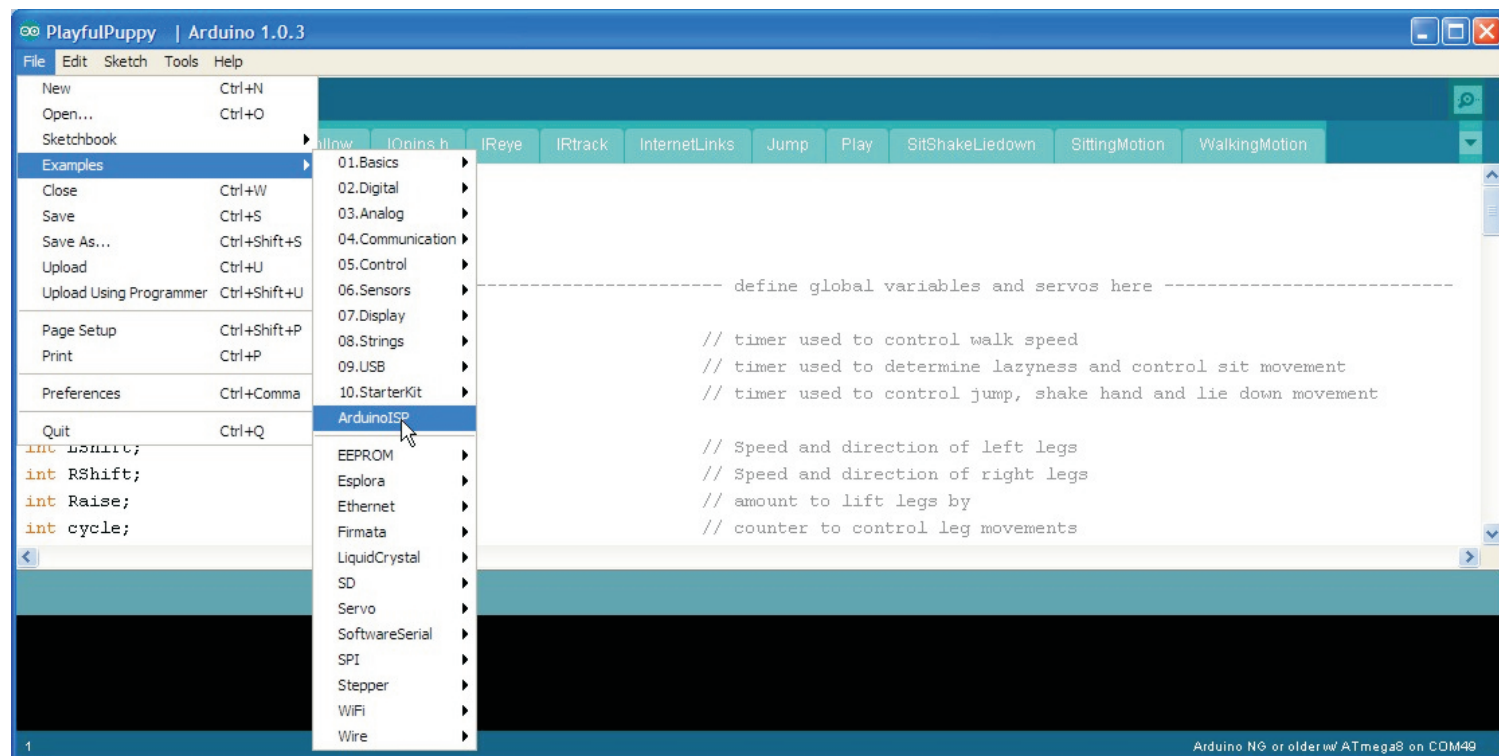
    if (tilt<1700) trick=millis();
    temp=millis()-trick;
    if (temp>600)
    {
        // reset timer if looking up
        // time how long head is looking down
        // if head is looking down for more than 600ms
    }
}
```


Uploading via the ISP socket:

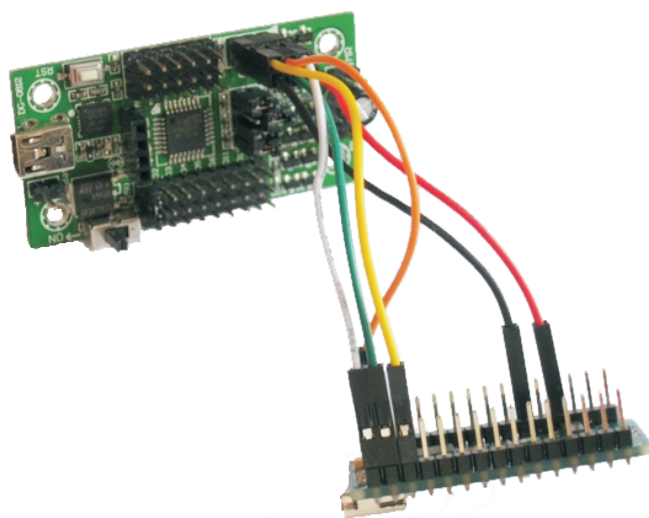
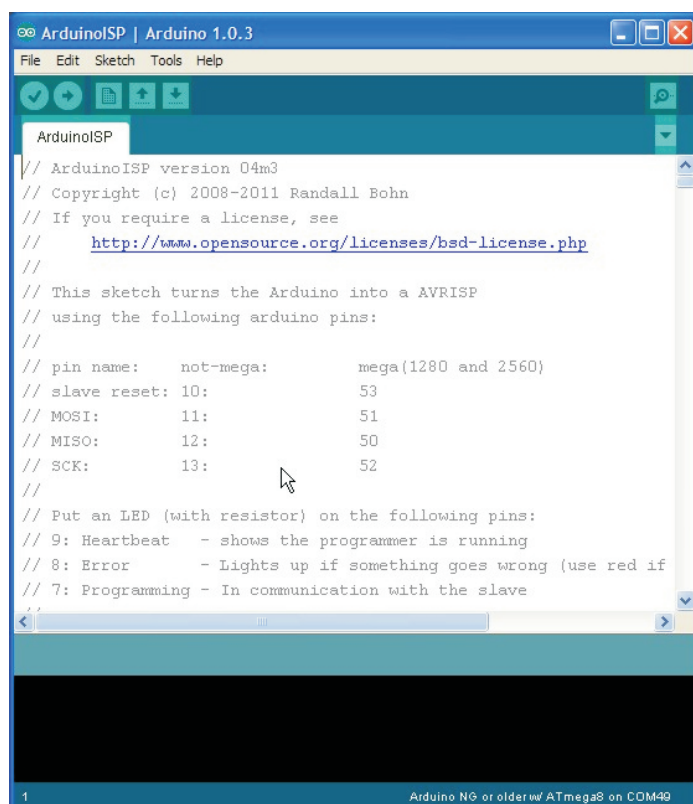
Since version 1.0 of the Arduino IDE it has been possible upload your programs directly to the processor using the ISP socket. This will overwrite the bootloader making more memory available and cause your program to start immediately on power up or reset.

To do this we need a programmer. If you have a programmer, please refer to it's instruction manual. These instructions will explain how to use an Arduino or compatible board as the programmer.

Using the example code supplied with the Arduino IDE you can use another Arduino or Arduino compatible board as a programmer. Go to the File menu, Examples and select **ArduinoISP**.



Upload this program into another Arduino or an Arduino compatible controller. This will then become your programmer. Use jumper wires to connect your programmer to the puppy robot's ISP socket as instructed in the programmer code.



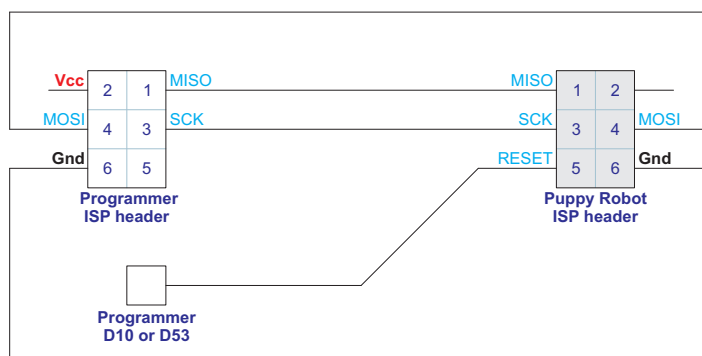
This photo shows an Arduino Nano being used as a programmer.

Uploading via the ISP socket (cont.)

You can connect the wires directly from the ISP socket of the programmer to the ISP socket of the puppy or you can use the digital pins. Note that D10 or D53 of the programmer is connected to the reset pin of the puppy.

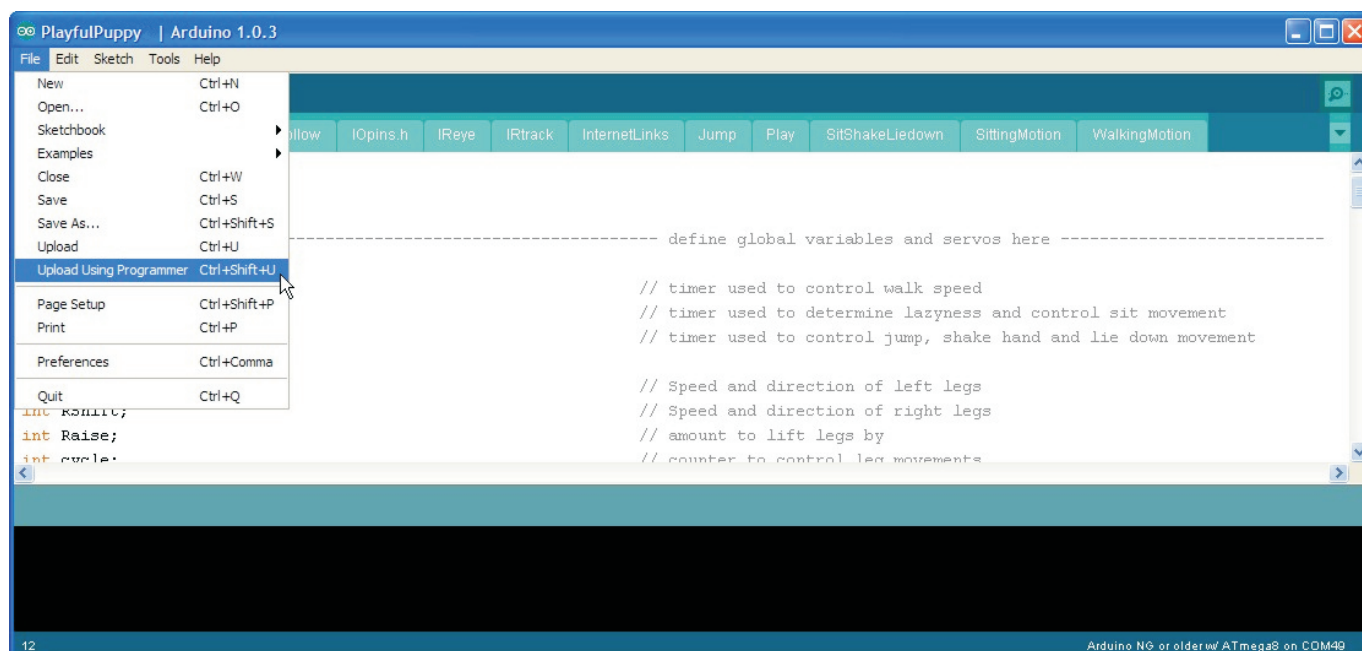
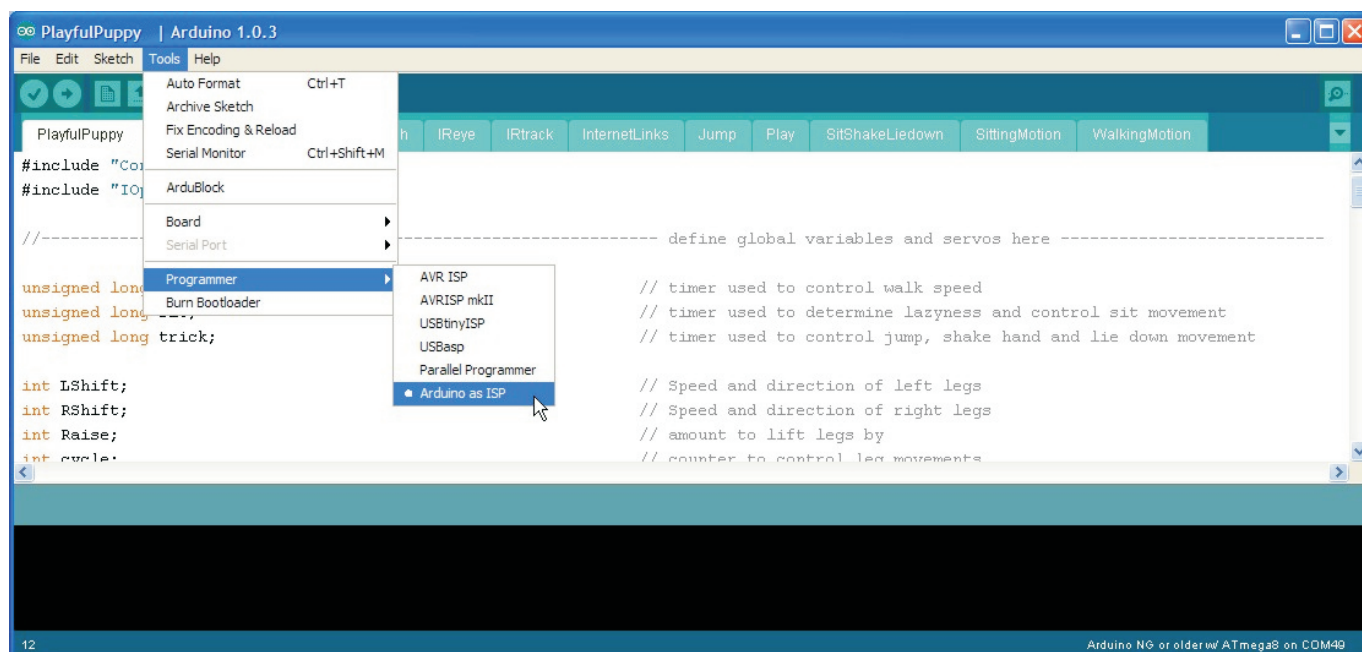
Configuration for boards using: ATmega 328 ATmega168 ATmega8	D12 / MISO	1	2	Vcc
	D13 / SCK	3	4	D11 / MOSI
	D10 or RESET	5	6	Gnd

Configuration for boards using: ATmega 2560 ATmega1280	D50 / MISO	1	2	Vcc
	D52 / SCK	3	4	D51 / MOSI
	D53 or RESET	5	6	Gnd



The programmer can be any Arduino compatible controller using an ATmega 8, 168, 328, 1280 or 2560 processor with a 5V logic supply. No matter what board you use for the programmer, leave the board type as Arduino NG or older w/ ATmega8 as this is the configuration of the puppy robot.

Go to the Tools menu, select programmer and choose your programmer type **Arduino as ISP**. Then select **Upload Using Programmer** from the file menu.



Uploading via the ISP socket (cont.)

For no apparent reason you may get an error message. If you try and upload a second time then the program should upload ok. If you continue to get this message then check your robot is turned on and that the batteries are fully charged. Check the wiring and make sure your board type and programmer type are set correctly. The serial port must be set to the programmer.

```
Error while burning bootloader.  
avrdude: stk500_getsync(): not in sync: resp=0x15
```

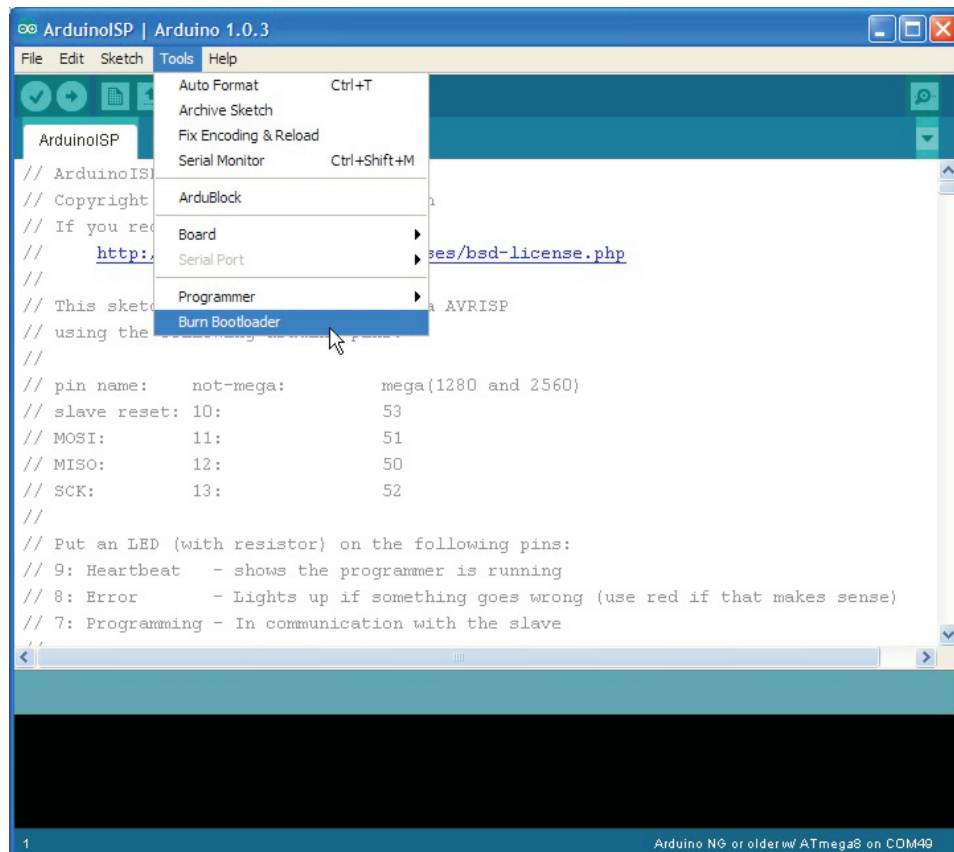
Burning the bootloader:

If you have been uploading code using the ISP socket and want to go back to using the USB port or your bootloader has been corrupted as the result of flat batteries then you will need to re-burn the bootloader.

The procedure is very similar to uploading code via the ISP socket. If you have not done so already, configure an Arduino or compatible board as your programmer as explained on page 21.

Once your programmer is prepared, go to the Tools menu and select your serial port to suit your programmer. Select the board type to Arduino NG or older w/ ATmega8. Now select **Burn Bootloader**.

As with programming via the ISP socket you may get an error message the first time. Select **Burn Bootloader** a second time and it should work fine.



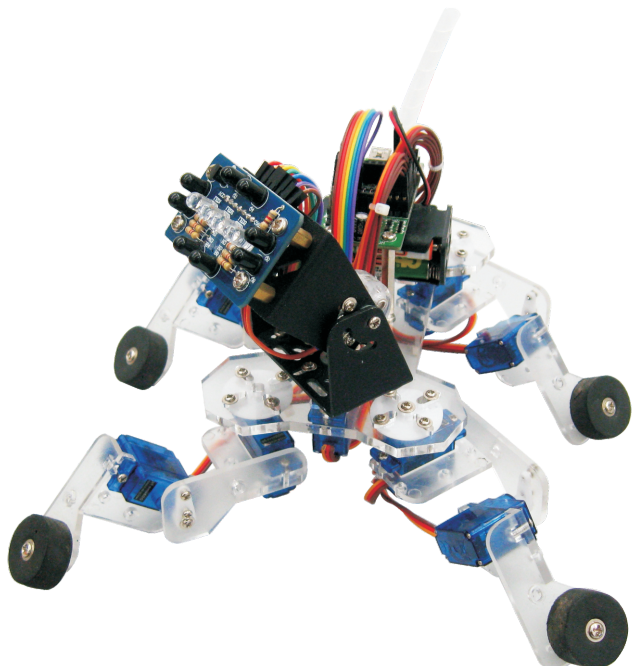
More detailed information on uploading code and burning the bootloader via the ISP socket can be found here: <http://letsmakerobots.com/node/35649>

[Letsmakerobots.com](http://letsmakerobots.com) is a site dedicated to the hobby of robot making. membership is free. You can ask questions on this site and get help from myself and other robot enthusiasts.

Trouble Shooting:

Please note that this robot is designed to work only with good quality, **AAA NiMh batteries**. Alkaline batteries will not be able to deliver the current necessary and should not be used. Do not use a 2S LiPo battery as the voltage is too high and may damage the servos.

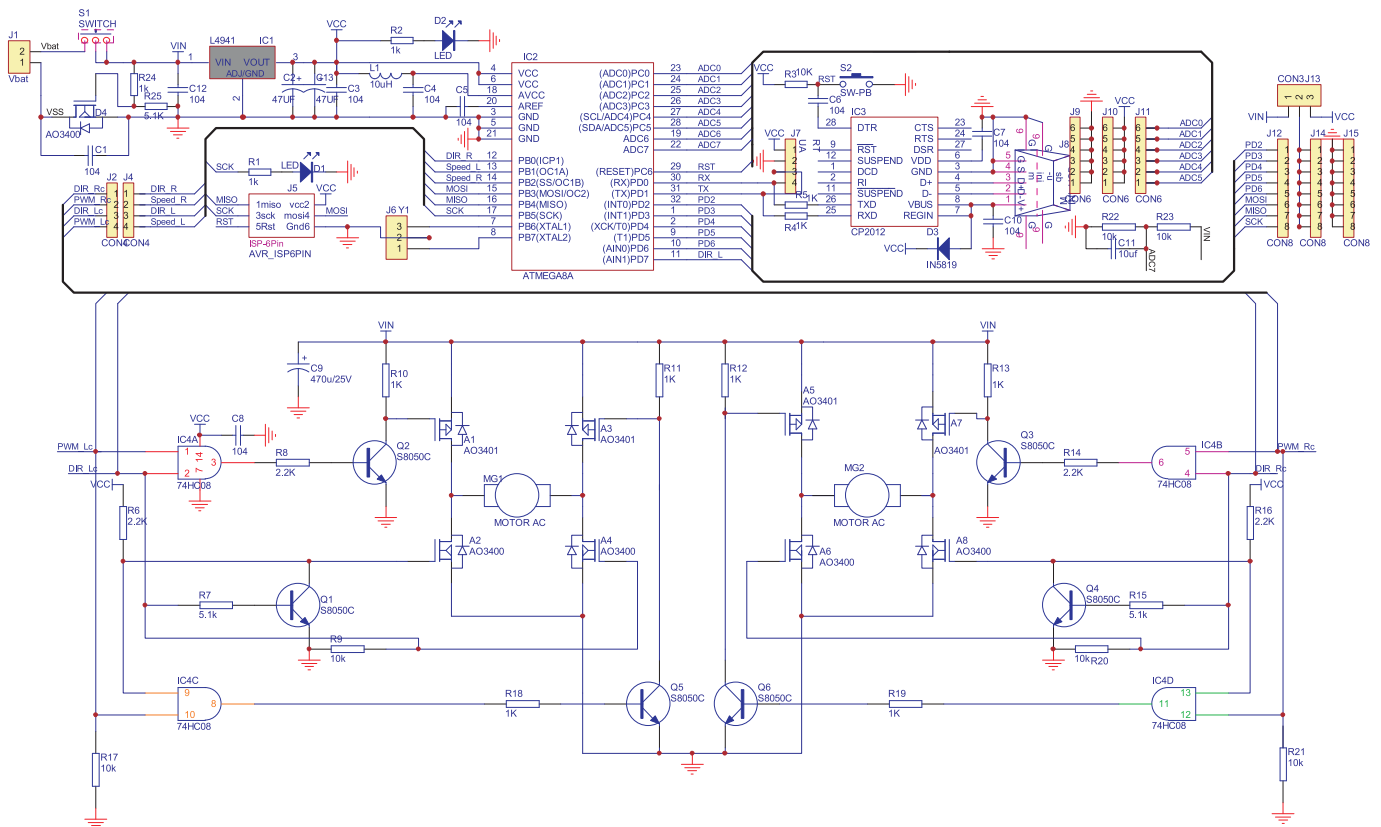
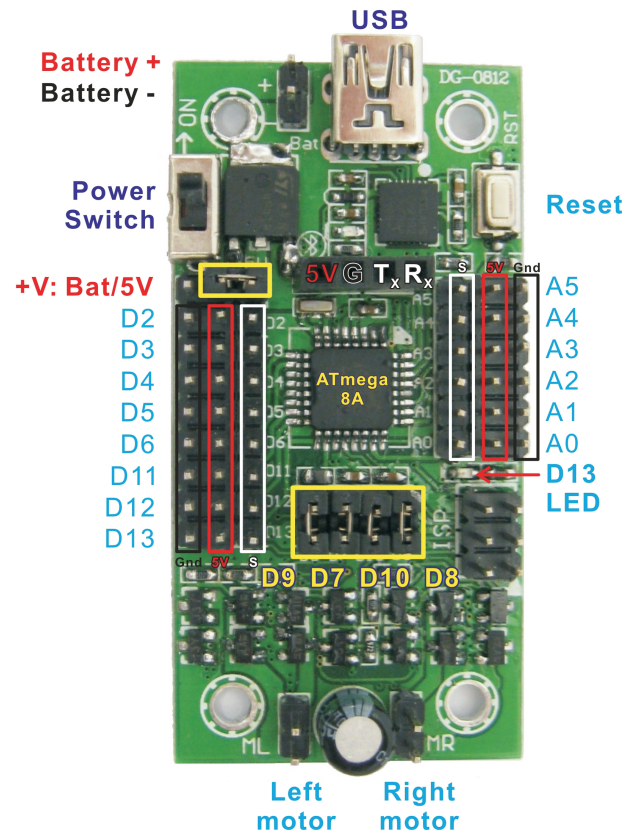
- Q. I turn the robot on but no lights come on.
A. Check that the batteries are fully charged and the connector is the correct way around with the red wire to +5V. Check all servos are connected with the ground wire (brown or black) to the outer edge of the PCB. Check the Vcc and ground wires for the compound eye are wired correctly.
- Q. The robot tries to stand up or move but then stops, waits a while and tries again.
A. The batteries are getting flat or the servo power selection jumper is in the wrong position. See page 9. The pan / tilt assembly is too tight causing the pan /tilt servos to overload the regulator.
- Q. I cannot upload the program.
A. Check that the correct board type and com port are selected. See page 11. Some USB ports may not be able to power the puppy during upload, turn the robot on. See page 12. Try a different USB cable. If the battery voltage got too low then it is possible for the bootloader to become corrupt. See pages 21 - 23. Check the com port is recognized and the correct port is selected. Try re-instaling the USB drivers.
- Q. The robot looks away from an object instead of towards it.
A. Check that the Pan/Tilt assembly has been assembled correctly. See page 7. Is the compound eye mounted upside down? See page 9. Check the wiring for the eye is correct.
- Q. The robot does not stand correctly / walk properly.
A. Enable the **return;** command at the start of the program loop so you can check the servo positions. This will freeze the program with all servos held at center position. Check that all servos are aligned properly. See page 5. Check that the hips are 90 degrees to the body and the knees are bent at 90 degree. See page 6. If the joints are out of alignment by a large amount then re-seat the servo horns. If the joint is out of alignment by a small amount then adjust the servo center position. See page 15. Check that you have your servos connected to the correct I/O pins. See page 10.
- Q. The head shakes when tracking an object.
A. If an object is too small then the head may have trouble locking on. Use an open hand instead of a fist. Try reducing values of **LRscalefactor** and **UDscalefactor** in the [**Constants.h**] tab. See page 15.
- Q. I have been experimenting with the code and now the head tracks objects too slowly.
A. As you add more code the speed at which the program loops may slow down. Try increasing values of **LRscalefactor** and **UDscalefactor** in the [**Constants.h**] tab. See page 15.



Robot Controller:

The Playful Puppy uses the **Mini Driver** which is a general purpose robot controller from DAGU.

This is a small, Arduino compatible controller that comes with the Arduino bootloader pre-installed.



Specifications:

- Input voltage 4.5V - 9V with reverse polarity protection rated at 4A.
- LDO 5V regulator rated at 1A.
- USB interface and ISP socket, can be programmed with USB cable or ISP programmer.
- ATmega8A processor @ 16MHz with 8K FLASH, 1K SRAM and 512 Bytes EEPROM memory.
- 8x digital I/O pins with servos compatible 3 pin male headers and selectable power (+5V / +Battery).
- 6x analog inputs with regulated 5V and Gnd for powering analog sensors.
- Built in battery monitor on analog input A7.
- Dual FET "H" bridge rated at 2.5A per channel can drive 2x DC motor or 1x Stepper motor.
- Serial port for optional Bluetooth transceiver.