Ringo Arduino Robot

Specs and coding

Me

Kent Archie

kentarchie@gmail.com

Code and slides on GitHub

https://github.com/kentarchie/RingoArduinoTalk

Plum Geek Robot History

Started as a Kickstarter project in Spring 2015

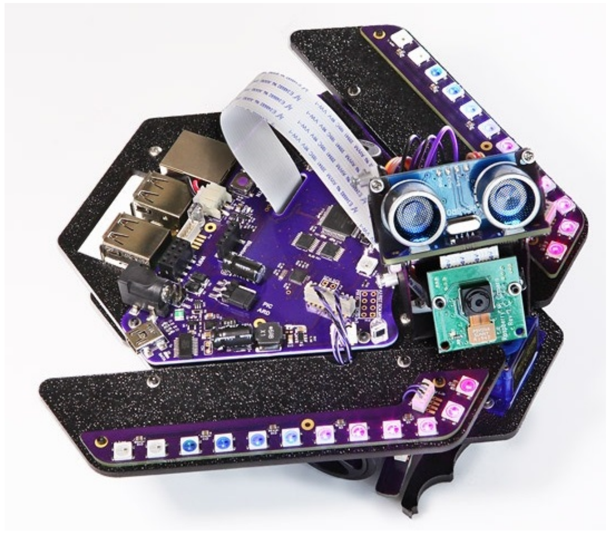
Delivered in September

Many interesting updates explaining the build process

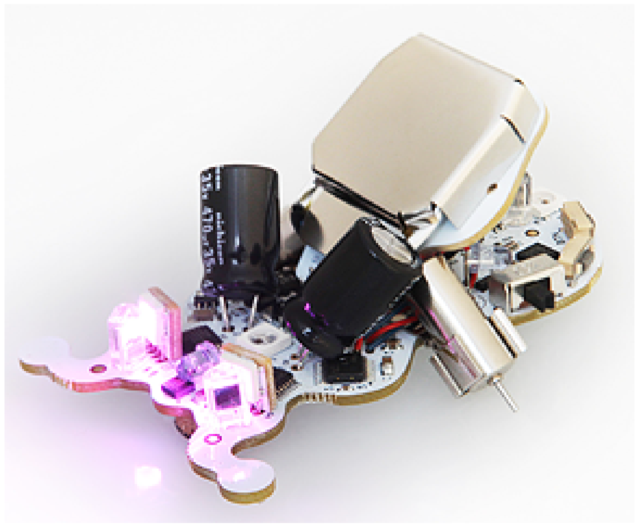
Company produces several small robots for educational and hobby purposes

Wink

Spirit



Ringo

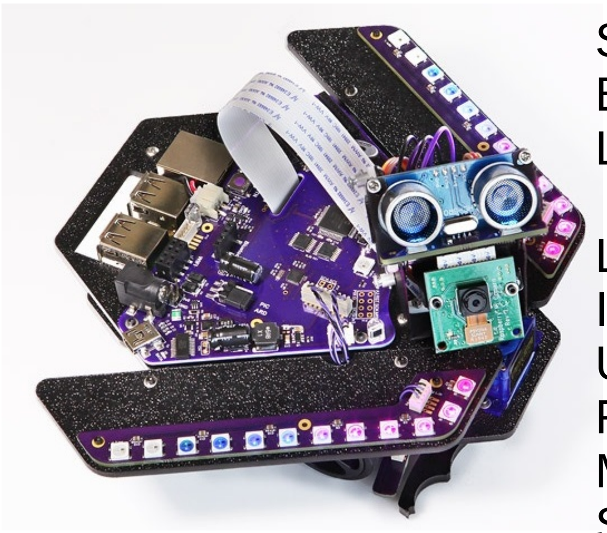


https://www.kickstarter.com/projects/plumgeek/ringo-the-palm-size-robot-with-real-personality

Other Robot Specs

Take your first steps into written code. Perfect for students who already know Scratch, and even those completely new to coding.

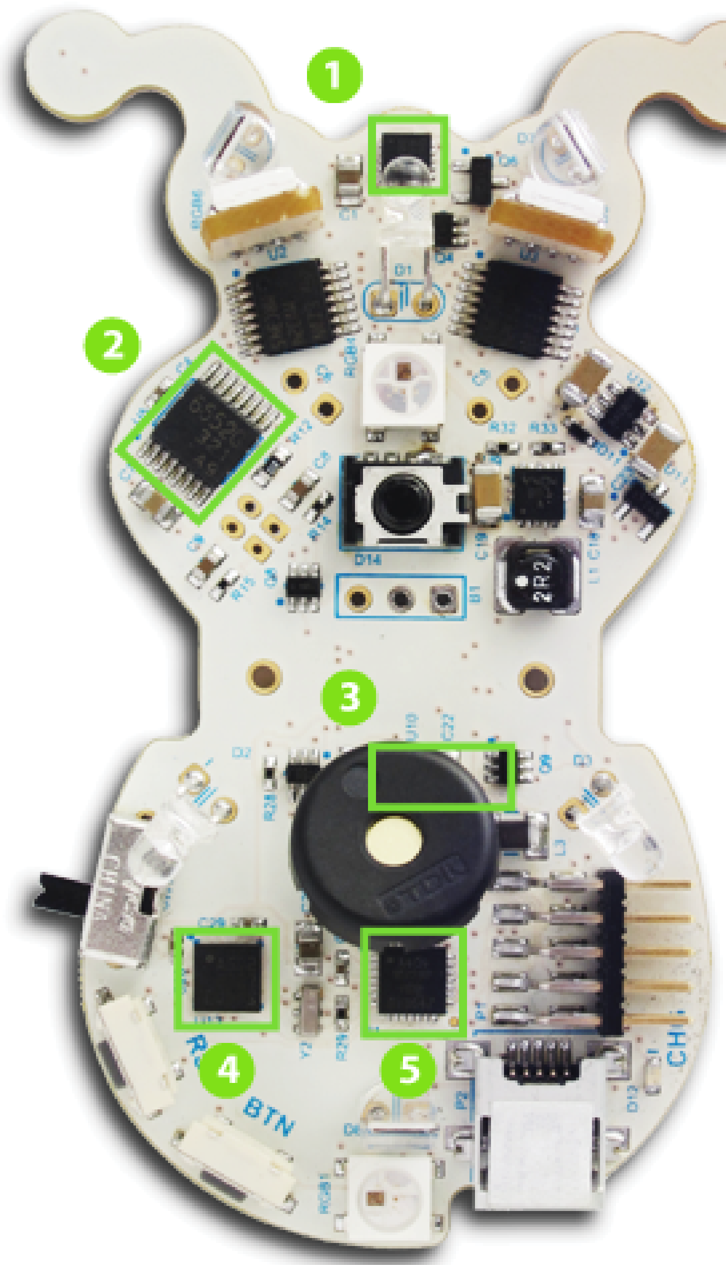
Teachers LOVE the excellent free Wink lesson series
No prior code experience necessary for students or teachers
Ideal for classroom instruction and workshops
Use free Arduino software for code
Basic smarts for light seeking, line following, and barrier detection
Suitable for ages 8+ and younger if working with a grownup



Spirit Rover   Skill Level 3
Built on Arduino and Raspberry Pi. This is a hacker's dream robot.
Learn Python, Linux, and Computer Vision.

Lessons teach a range of basic to advance robotics concepts.
Ideal for classroom instruction and workshops
Use Arduino, Python, and Linux
Remote control & program over the internet, WiFi, or smart phone
Many advanced smarts and configuration options
Suitable for ages 12+ and younger if working with a grownup

Ringo Robot Specs

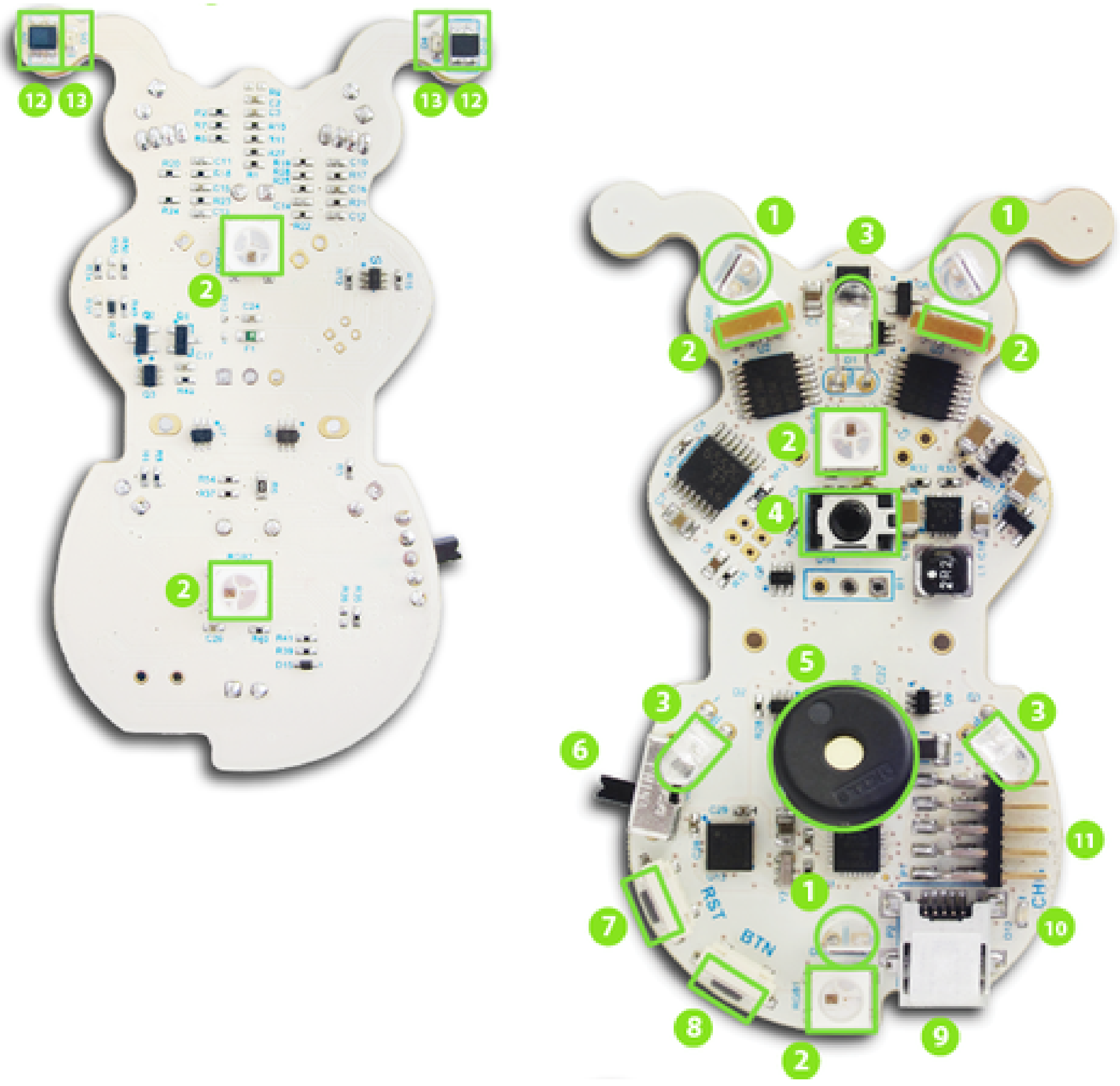


Smart Parts

1. Accelerometer
2. Motor Driver
3. Battery Charger
4. Gyroscope
5. Arduino UNO MCU

Ringo images and text from
http://www.plumgeek.com/ringo.html

Ringo Robot Inputs



Ins and Outs

1. Ambient Light Sensor
2. RGB NeoPixel LED
3. IR LED Transmitter
4. 38kHZ TV Remote Receiver
5. Piezo Sound Element
6. Power Switch
7. Reset Button
8. User Button
9. USB Port for Charging
10. Charging Status LED
11. Programming Port
12. Edge/Line Sensor
13. IR LED Surface Illuminator

Component Details

3-Axis Accelerometer:
Can be used to sense movement in all 3 axis.
Can also be used to determine the orientation of the robot as well as tap detection.
So you can trigger behaviors when the robot is moved or swatted by your cat.
This sensor can be used as a basis to make Ringo travel specific distances, or when moved by an outside force to return to its starting position, etc.
The part is a Freescale MMA8451QR1.

3-Axis Gyroscope:
Can be used to determine which direction Ringo is facing.
It works in all 3 axis, though the flat axis will be most useful as you can determine how far it has turned, or to cause it to turn a specific number of degrees.
The part is an ST L3GD20TR.

Six RGB LEDs:
Each of these LEDs can create a mix of red, green, and blue which basically allows you to create any color you want by mixing the three values. These are the same lights used in NeoPixel products from Adafruit Industries,
The code to control the lights is super simple to use and the lights enable all kinds of expression.
Two lights on the top, two on the bottom, and both eyes light up.

Component Details

Sound Element:
The piezo sound element can create any sort of pings, tones, and chirps you can think of. Can be used to give Ringo a voice as it explores its environment. Can also be used to play musical notes.

Light Sensors:
Three sensors can measure ambient light 360 degrees around Ringo, allowing it to respond to light, shadows, etc.

Infrared Light Sources:
Three IR light sources are placed pointed 120 degrees apart. They can be enabled individually in any pattern and can be driven together at the same time. This allows creating signals like those from a TV remote to communicate with other Ringo bots or control appliances like your TV. They can also be used together with the Light Sensors to detect objects or movement near Ringo.

38 kHz Receiver:
This is a special sensor designed to sense the modulated light signal produced by most TV remote controls. Use the included remote or teach your Ringo to respond to your own remote.

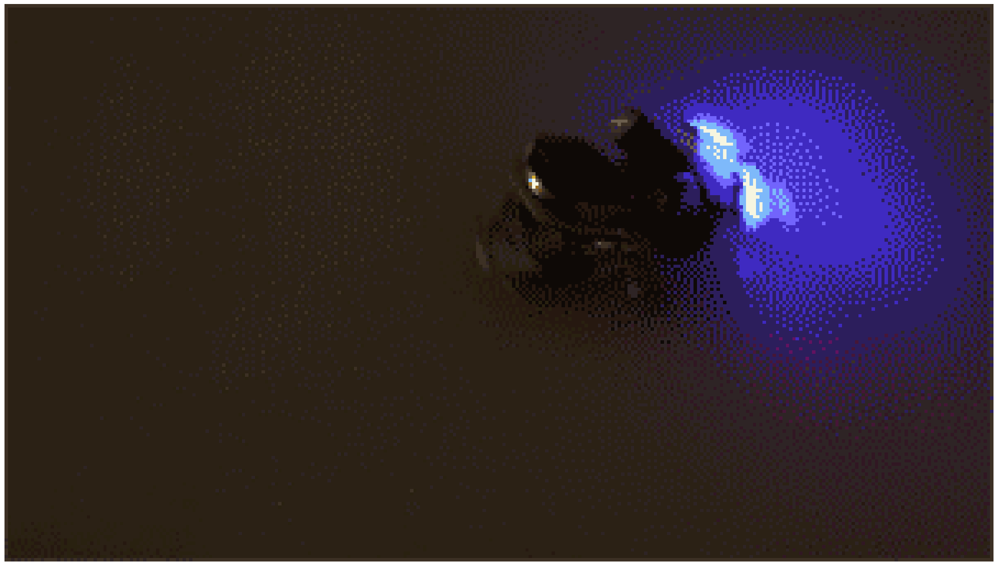
Component Details

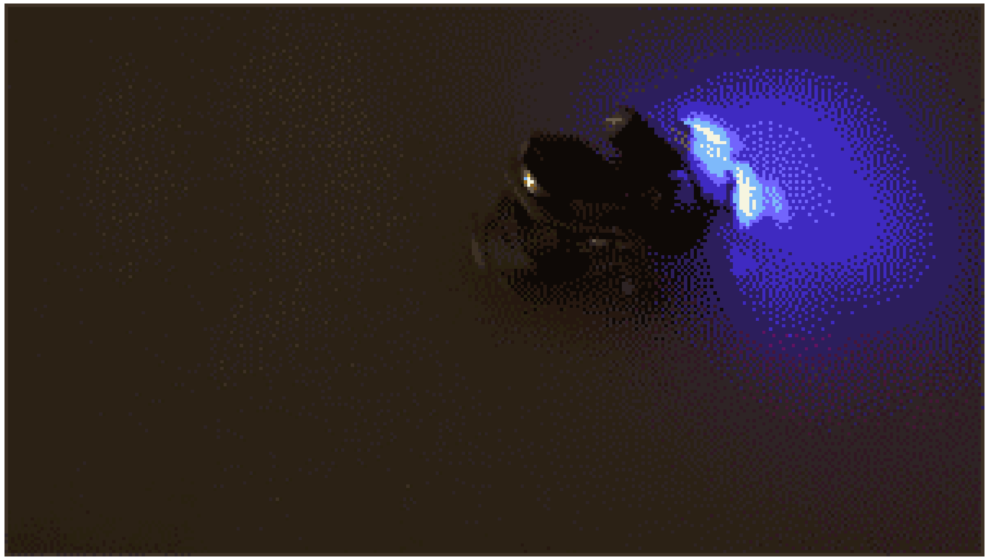
Edge Sensors:
A light sensor and an IR light source are hidden under each of Ringo's feelers and also at the rear of Ringo. This allows it to sense edges and follow lines. We've had success creating mazes with lines or tape on a desk.

Charging:
Ringo charges its battery automatically whenever plugged into the programming adaptor, or when a USB cable is connected directly to its USB port. (Note the USB port on Ringo does not provide any data communication, it is simply used as charging input). A "charging status" LED is provided so you can see when it has completed charging.

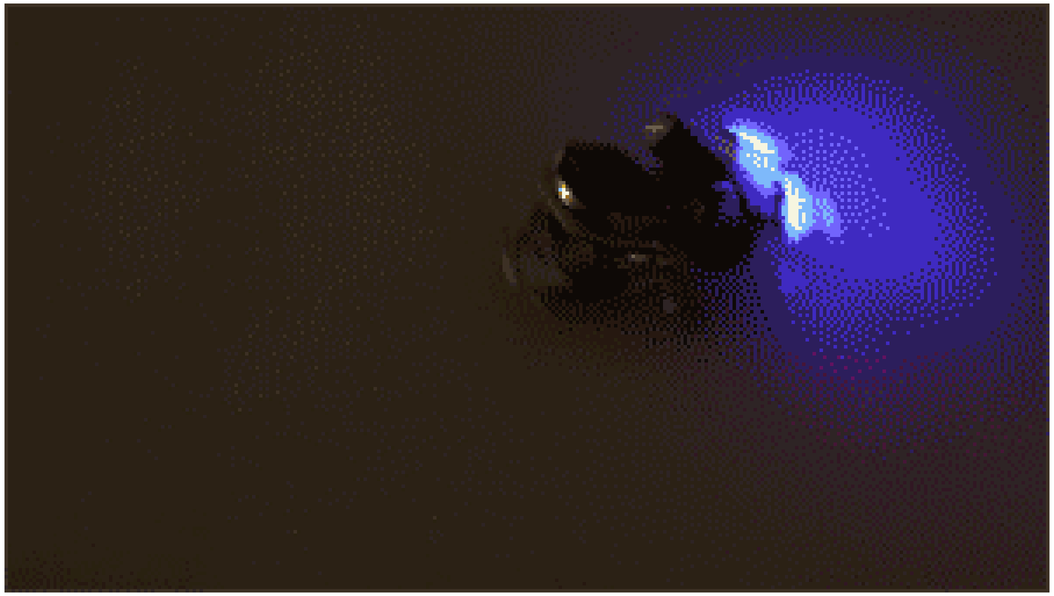
Pager Motors:
The tips of the motors contact the running surface, allowing Ringo to skate around your table. Both motors can be controlled independently. They can run forward or backward at variable speed.

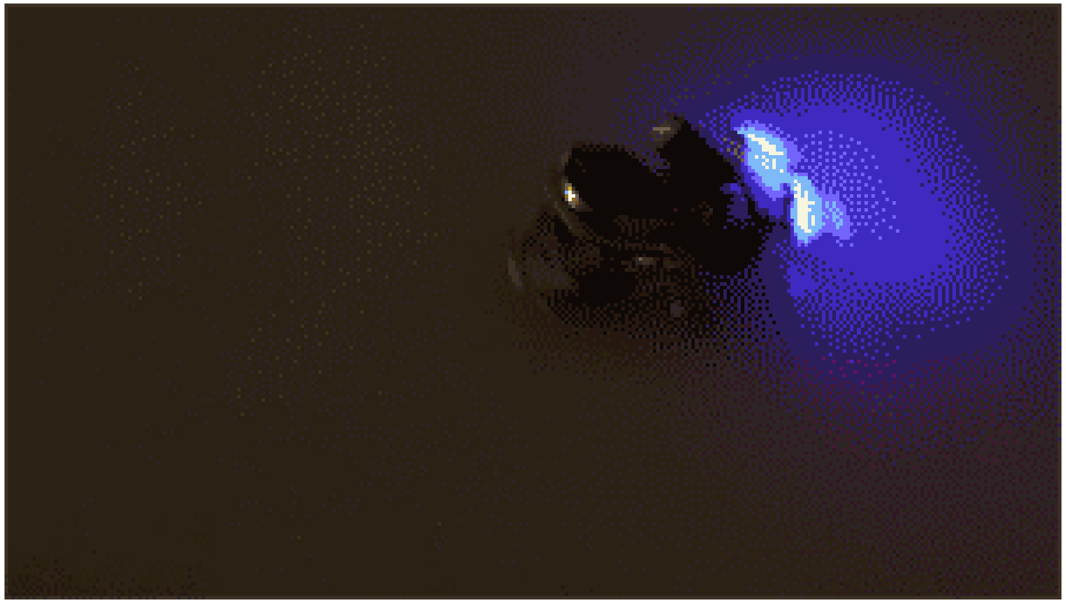
Ringo Action Examples



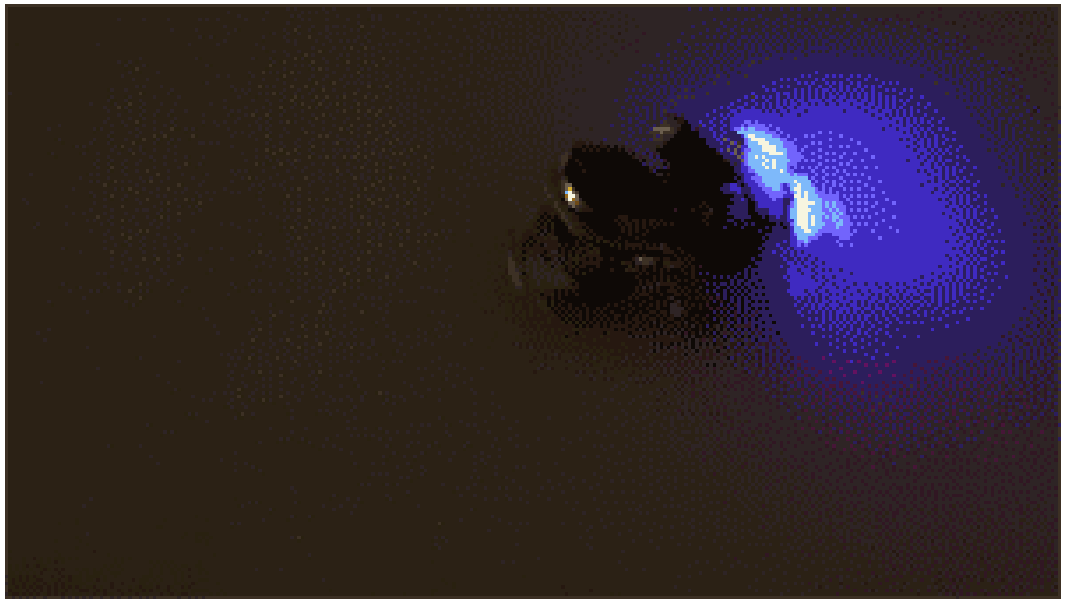


Ringo Action Examples

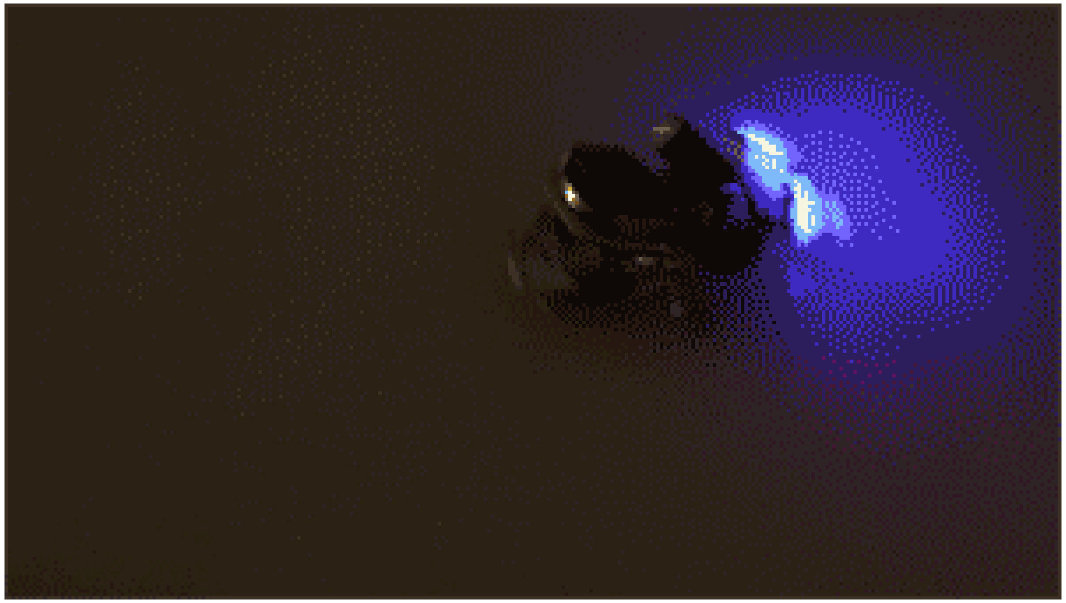

Barrier Detection


Edge Detection

Ringo Action Examples



Light Detection



Navigation

Arduino

Information and IDE available here https://www.arduino.cc/

Microcontroller	ATmega328P
Operating Voltage	5V
Input Voltage (recommended)	7-12V
Input Voltage (limit)	6-20V
Digital I/O Pins	14 (of which 6 provide PWM output)
PWM Digital I/O Pins	6
Analog Input Pins	6
DC Current per I/O Pin	20 mA
DC Current for 3.3V Pin	50 mA
Flash Memory	32 KB (ATmega328P) of which 0.5 KB used by bootloader
SRAM	2 KB (ATmega328P)
EEPROM	1 KB (ATmega328P)
Clock Speed	8 MHz

There are a lot of different styles and versions of Arduino, this is the one Ringo uses

Arduino Vs Raspberry Pi

The Arduino is used in embedded operations. There isn’t a regular OS.
Your application is the only thing it runs

The Raspberry Pi is a proper computer, running a version of Linux.

A good, short discussion of this is available here

https://makezine.com/2015/12/04/admittedly-simplistic-guide-raspberry-pi-vs-arduino/

Arduino IDE

Information and IDE available here https://www.arduino.cc/

The Ringo instructions are pretty simple.
They give a list of things to get and install, including the IDE, a USB driver
and the Ringo libraries

Arduino programs are called sketches and each sketch is in its own directory.
Plum Geek provides a base sketch that includes all the needed libraries but
doesn’t actually do anything

The Ringo libraries hide the low level programming details,
but all the code is open source so you can change any of it

I built the example code using the Windows desktop IDE.
Once it is installed, there are command line tools you can use.
You can also specify an external editor so you don’t have to use theirs.
I’m a Vim user, so I used the external editor option but did builds using the IDE.

I saw some examples of people using a standard editor and make to build and
Install the programs.

Arduino IDE

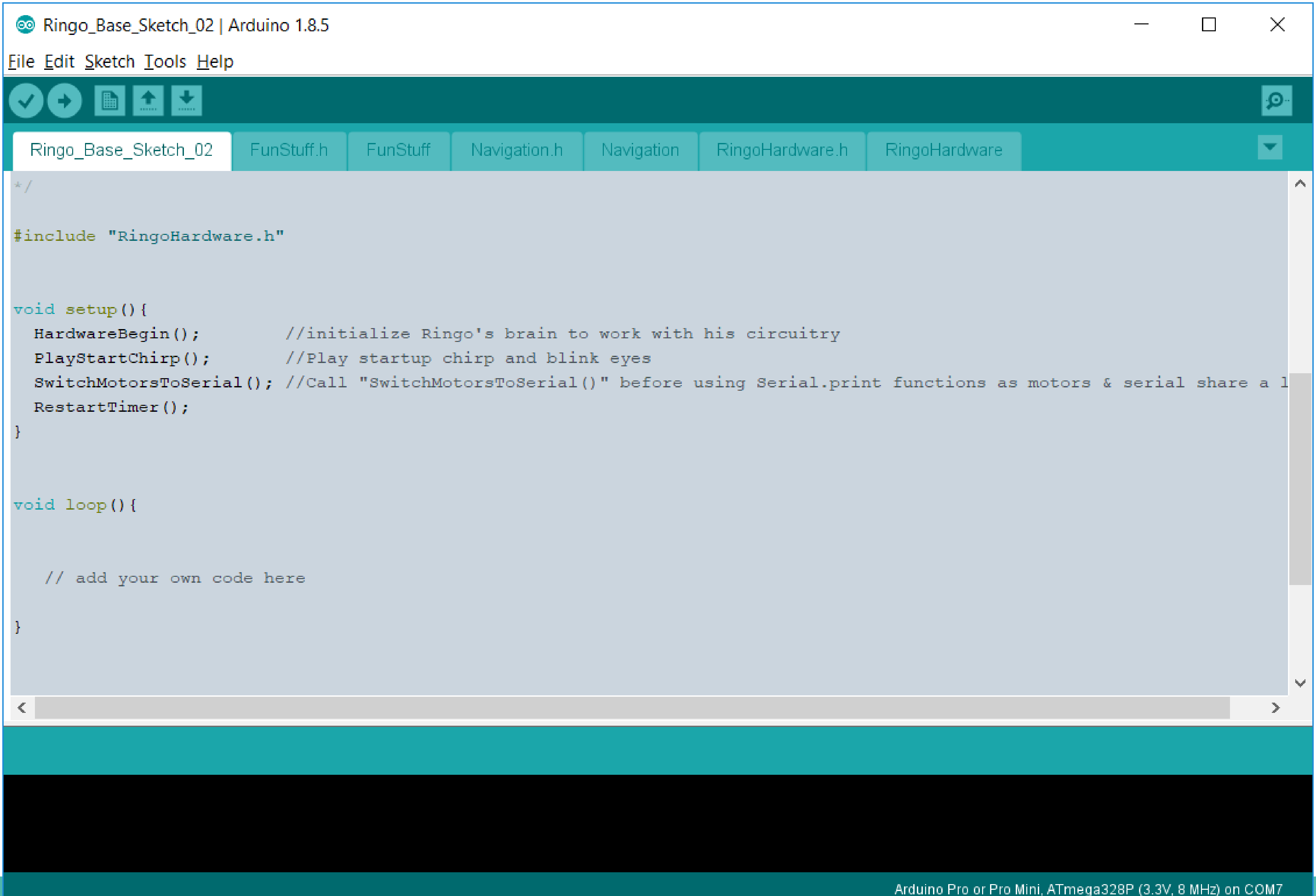
Mainly Arduino programming is done in C/C++.
The C code is in files that end in .ino rather than .c or .cpp
The compiler combines the .ino files together with less work on the developers part,
But you can use the traditional file endings.

The IDE has only a few commands
Build (called verify)
Upload (transfer executable to the device)
Serial Monitor (monitor data going over the connection)

The Serial Monitor is how you see the results of print statements in the code

You have to tell the IDE which Arduino device you are using.

Arduino IDE Basic Sketch



```
*/

#include "RingoHardware.h"


void setup(){
  HardwareBegin();        //initialize Ringo's brain to work with his circuitry
  PlayStartChirp();       //Play startup chirp and blink eyes
  SwitchMotorsToSerial(); //Call "SwitchMotorsToSerial()" before using Serial.print functions as motors & serial share a l
  RestartTimer();
}


void loop(){


   // add your own code here

}
```

Arduino Pro or Pro Mini, ATmega328P (3.3V, 8 MHz) on COM7

Arduino Coding

There are two required functions in an Arduino program

void setup() {}

This is code that is executed when the device is powered up.
It runs once.
Initializations go here

void loop() {}

This runs repeatedly and forever
Simpler method for beginners and amateurs
Can make some code clumsy
You can make interrupt handlers for some events

Getting Light Values

```
int GetRearLightSensor()
{
   SetPixelRGB(TAIL_TOP,0,0,0); // turn off back light
   int lightValue = analogRead(LightSense_Rear); // read Rear light sensor
   return(lightValue);
} // GetRearLightSensor
```

Flashing Lights

```
void LightsOnLoop()
{
  for (int pixel = 0 ; pixel < NUM_PIXELS; ++pixel) {
      if((AreHeadLightsOn && ((pixel == EYE_LEFT) || (pixel == EYE_RIGHT))))
         continue;
      SetPixelRGB( pixel, NEW_RED, NEW_GREEN, NEW_BLUE);
      delay(PIXEL_FLASH_DELAY);
      OffPixel(pixel);
      delay(PIXEL_DELAY);
  }
} // LightsOnLoop
```

Move Towards The Light

```
void FollowLight()
{
   int leftSensorValue = 0, rightSensorValue = 0;
   OffEyes(); // need front lights off to take readings
   printString("Eyes Off");

   leftSensorValue  = analogRead(LightSense_Left);
   printValue("FollowLight: Left Sensor = ", leftSensorValue);
   rightSensorValue = analogRead(LightSense_Right);
   printValue("FollowLight: Right Sensor = ", rightSensorValue);

      printString("FollowLight: head light check reset");

   if((leftSensorValue > InitialAverageLight) &&
      ((leftSensorValue - InitialAverageLight ) > TooDarkValue))
      {
         printString("Turning Left 90 degrees");
         RotateAccurate(-90, 1500);
         SwitchMotorsToSerial(); //Call before using Serial.print functions motors & serial share line

         printString("Left move complete");
         PlayLeftTune();
         printString("Left tune complete");
      }
} // LightsOnLoop
```

Play A Tune

```
void PlayLeftTune()
{
   PlaySweep(4000, 1000, 330);
   //OffChirp();
} // PlayLeftTune
```

Setup

```
void setup()
{
   Serial.begin(57600);
   while (! Serial); // Wait until Serial is ready - Leonardo

   HardwareBegin();        //initialize Ringo's brain to work with his circuitry
   SwitchButtonToPixels(); //set shared line to control NeoPixel lights
   PlayStartChirp();       //Play startup chirp and blink eyes
   RestartTimer();
   delay(1000);
   NavigationBegin();

   GetInitialLightValue();

   printString("Setup Done");
} //setup
```

Loop

```
int lightSensorValue = 0;
   LoopCounter++;
  // delay(LIGHT_SENSE_DELAY);

   LightsOnLoop();

   digitalWrite(Source_Select, HIGH); //This selects the top light sensors
   lightSensorValue = GetRearLightSensor();

   printValue("Rear Light Sensor = ", lightSensorValue);

   HeadLightsOn(lightSensorValue);
   FollowLight();

   printString("Loop Done");
   /*
   if(LoopCounter > 30) {
      printString("Ringo Shutdown");
      exit(0);
   }
```

But I don’t Like C

There are several other development choices and different IDEs

Once the desktop IDE is installed, you can call the compiler and
uploaded from the command line

There is at least one plugin that allows use of Visual Studio

There are a couple of graphical coding tools to build Arduino programs

You can develop code in C# and Python

I found some references to coding in Basic and Forth

There is a web based IDE