Roomba Hacking …with Ruby!

a short story by Eric Wood

illustrations by Google Image Search

eric@ericwood.org | @eric_b_wood

what is a roomba

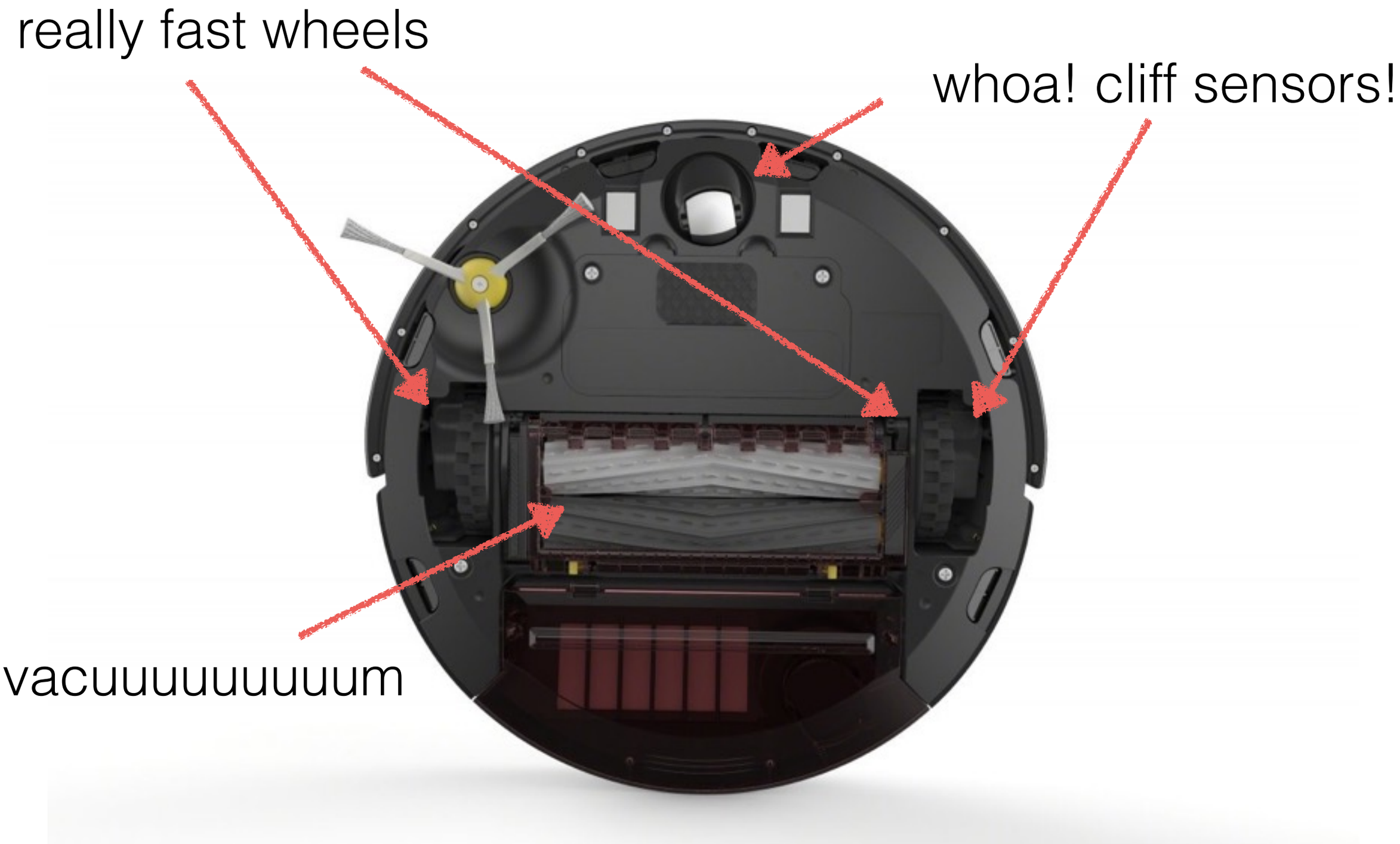
buttons and
neato lights

crazy powerful
brushes of doom

IR sensor! whoa!

insanely cool bumper sensors

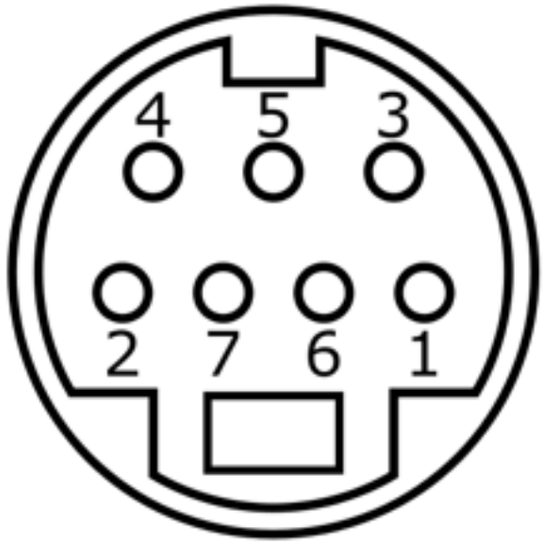
what is a roomba



so…why?

supplies

- Roomba!
- Special cable!
  - (serial —> 7-pin mini-DIN)
- Some kind of computer?



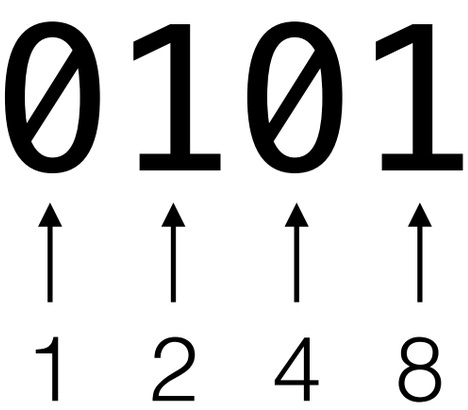
Roomba SCI Primer!

the format!

- opcodes
- magic numbers for each command
- they can take arguments (data bytes)
- 8-bit unsigned integer!

let's talk binary real quick…

the number 10 in binary:



Unsigned! (no notion of positive/negative)

Big-endian! (most significant bit is at the end!)

let’s talk binary real quick…

byte — 8 bits

In Ruby:

Fixnum:

```
>> 1337.size
=> 8
```

Bignum:

```
>> 100000000000000000000000000000000.size
=> 13
```

our first opcode!

Start

Opcode: 128

Data bytes: none

(must be sent before ANYTHING else!)

modesssssss

- Off
  - zzzzzzzzzzzzzz
- Passive
  - We can control anything as long as it doesn’t move (AKA “boring mode”)
- Safe
  - Actuator control, safety-related sensors engaged (wheel drop, cliff, etc.)
- Full!!!!!!
  - Do whatever you want, but you have to deal with the consequences (AKA “fun mode”)

mode-setting opcodes

- We have to set the mode now!
  - **131** safe
  - **132** full
  - **133** power (virtual press of power button)

how about some motors?

137: drive

4 data bytes (16 bit, signed, twos-complement)

```
[velocity (2 bytes)][radius (2 bytes)]
```

Velocity: speed in mm/s (positive: forward, negative: backwards)

Radius: turning radius (mm/s) (positive: left, negative: right)

driving example

Goal: drive in reverse at 200mm/s
while turning at a radius of 500mm

`Velocity = -200 = 0xFF38 = [0xFF][0x38]`

`Radius = -500 = 0x01F4 = [0x01][0xF4]`

Bytes sent:
`[137][255][56][1][244]`

`[0x89][0xFF][0x38][0x01][0xF4]`

sensorsssssssss

- Bump (front bumper, left and right)
- Wheel drops
- Cliff (left, right, front left, front right)
- Virtual wall
- Dirt (left, right)
- Motor overcurrents (each motor type)
- Remote control (button presses, etc.)
- Buttons
- Wheel rotation (angle, distance)
- Temperature (sure, why not?
- MORE THINGS THAT NOBODY CARES ABOUT

requesting data

- Query by grouping (lame)
- Query list (pick which sensors!)
- Stream (query list, but no polling)

sensor packet groups

Packet Membership								Name	Bytes	Value Range	Units
0	1					6	7	Bumps and Wheel Drops	1	0 - 31	
							8	Wall	1	0 - 1	
							9	Cliff Left	1	0 - 1	
							10	Cliff Front Left	1	0 - 1	
							11	Cliff Front Right	1	0 - 1	
							12	Cliff Right	1	0 - 1	
							13	Virtual Wall	1	0 - 1	
							14	Overcurrents	1	0 - 31	
							15	Unused	1	0	
							16	Unused	1	0	
		2					17	IR Byte	1	0 - 255	
							18	Buttons	1	0 - 15	
							19	Distance	2	-32768 - 32767	mm
							20	Angle	2	-32768 - 32767	mm
			3				21	Charging State	1	0 - 5	
							22	Voltage	2	0 - 65535	mV
							23	Current	2	-32768 - 32767	mA
							24	Battery Temperature	1	-128 - 127	degrees Celsius
							25	Battery Charge	2	0 - 65535	mAh
							26	Battery Capacity	2	0 - 65535	mAh

query list

opcode: **149**
arguments: # of packets, packet IDs

example:
get distance travelled (19) and bumper status (7)

[149] [2] [19] [7]

RUBY TIME

speaking serial

```ruby
require 'serialport'

# port is very OS/driver dependent...
# Typically, on *nix you're want this:
port = '/dev/ttyusbserial'

# baud:
# 115200 for Roomba 5xx
# 57600 for older (and iRobot Create)
baud = 115200
@serial = SerialPort.new(port, baud)
@serial.write('hello!')
```

implem

```c
/* courtesy of ruby-serialport/ext/native/serialport.h */
#include <ruby.h>    /* ruby inclusion */
#ifdef HAVE_RUBY_IO_H      /* ruby io inclusion */
   #include <ruby/io.h>
#else
   #include <rubyio.h>
#endif
```

changing formats

`Array#pack` to the rescue!

"Packs the contents of arr into a binary sequence according to the directives in a *TemplateString*"

```
Integer      | Array   |
Directive    | Element | Meaning
---------------------------------------------------------------------------
   C         | Integer | 8-bit unsigned (unsigned char)
   S         | Integer | 16-bit unsigned, native endian (uint16_t)
   L         | Integer | 32-bit unsigned, native endian (uint32_t)
   Q         | Integer | 64-bit unsigned, native endian (uint64_t)
             |         |
   c         | Integer | 8-bit signed (signed char)
   s         | Integer | 16-bit signed, native endian (int16_t)
   l         | Integer | 32-bit signed, native endian (int32_t)
   q         | Integer | 64-bit signed, native endian (int64_t)
```

pack example

opcode definition:
“8 bit unsigned integer”

`Array#pack` directive: `"C"`

```
[128].pack('C')
#=> "0x80"
```

our write function

```ruby
# Converts input data (an array) into bytes before
# sending it over the serial connection.
def write_chars(data)
  data.map! do |c|
    if c.class == String
      result = c.bytes.to_a.map { |b| [b].pack("C") }
    else
      result = [c].pack("C")
    end

    result
  end

  data = data.flatten.join

  @serial.write(data)
  @serial.flush
end
```

back to driving!

```
# Convert integer to two's complement signed 16 bit integer (big endian)
def convert_int(int)
  [int].pack('s>')
end
```

```
def drive(velocity, radius)
  raise RangeError if velocity < -500 || velocity > 500
  raise RangeError if (radius < -2000 || radius > 2000) && radius != 0xFFFF

  velocity = convert_int(velocity)
  radius   = convert_int(radius)
  write_chars([DRIVE, velocity, radius])
end
```

reading sensor data

```ruby
# Get sensors by list
# Array entry can be packet ID or symbol
def get_sensors_list(sensors)
  # request sensor data!
  request = [Constants::QUERY_LIST, sensors.length] + sensors
  write_chars(request)

  raw_data = ""
  sensors.each do |id|
    raw_data << @serial.read(SENSOR_PACKET_SIZE[id])
  end

  sensor_bytes_to_packets(raw_data, sensors)
end
```

handling sensor data

```
# Convert sensors bytes to packets hash
def sensor_bytes_to_packets(bytes, packets)
  # template string for unpacking the data
  pack = ''
  packets.each do |packet|
    size = SENSOR_PACKET_SIZE[packet]
    signedness = SENSOR_PACKET_SIGNEDNESS[packet]
    case size
    when 1 # 8 bit (big endian)
      pack << signedness == :signed ? 'c' : 'C'
    when 2 # 16 bit (big endian)
      pack << signedness == :signed ? 's>' : 'S>'
    end
  end

  data = bytes.unpack(pack)

  # snip...
end
```

bitmasks

Bumps and Wheel Drops **Packet ID: 7** **Data Bytes: 1**

unsigned

The state of the bumper (0 = no bump, 1 = bump) and wheel drop sensors (0 = wheel raised, 1 = wheel dropped) are sent as individual bits.

Range: 0 - 31

Bit	7	6	5	4	3	2	1	0
Sensor	n/a	n/a	n/a	Wheeldrop Caster	Wheeldrop Left	Wheeldrop Right	Bump Left	Bump Right

```
class BumpsAndWheelDrops
  def self.convert(v)
    h = {}
    h[:bump_right]       = v & 0b0001 > 0
    h[:bump_left]        = v & 0b0010 > 0
    h[:wheel_drop_right] = v & 0b0100 > 0
    h[:wheel_drop_left]  = v & 0b1000 > 0
    h
  end
end
```

making a “DSL”

goal

```
require 'rumba'

Roomba.new('/dev/tty.usbserial') do
  safe_mode
  forward 1.meter
  rotate :left
  rotate -90 # degrees

  rotate :right
  rotate 90
  backward 1.meter

  # access to any methods in the Roomba class here!
end
```

high-level methods

```ruby
# move both wheels at the same speed in a certain direction!
# NOTE THAT THIS BLOCKS UNTIL COMPLETE
def straight_distance(distance, speed: DEFAULT_SPEED)
  total = 0
  straight(speed)
  loop do
    total += get_sensor(:distance).abs
    break if total >= distance
  end

  halt
end

# distance is in mm!
def forward(distance, speed: DEFAULT_SPEED)
  straight_distance(distance, speed: speed)
end

# distance is in mm!
def backward(distance, speed: DEFAULT_SPEED)
  straight_distance(distance, speed: -speed)
end
```

patch them monkeys!

```ruby
# MEASUREMENT HELPERS
class Fixnum
  def inches
    25.4 * self
  end
  alias_method :inch, :inches

  def feet
    self.inches * 12
  end
  alias_method :foot, :feet

  def meters
    self * 1000
  end
  alias_method :meter, :meters
end
```

(btw this is important…)

```
# when the program is killed, stop the Roomba
trap("INT") do
  self.halt
  self.stop_all_motors
  self.power_off
  exit
end
```

might as
well demo
some stuff?

questions?!

learn more!

- Rumba gem
- Official iRobot SCI documentation
- DIY Roomba serial cable
- Order a custom-painted Roomba (very useful)