Distributed **Robots** in **Elixir**

http://isotope11.com

Josh Adams

/knewter

/knewter

http://elixirsips.com

Robby Clements

/robby_clements

/rclements

/rclements/erlang_factory_robots

What you'll get out of this talk

Robots; Android + Erlang + Elixir interop; Save days of frustration

This talk has two parts:

Survey / Knowledge Dump
Project Narrative

Part the first:

Survey of Robotics with Elix

Robots?

- A machine capable of carrying out a complex series of actions automatically.
- (esp. in science fiction) A machine resembling a human being and able to replicate certain human movements and functions.
- Robotics: The branch of technology that deals with the design, construction, operation, and application of robots.

Robots?

- A machine capable of carrying out a complex series of actions automatically.
- (esp. in science fiction) A machine resembling a human being and able to replicate certain human movements and functions.
- Robotics: The branch of technology that deals with the design, construction, operation and application of robots.

ROBOTS!!!1!!!one

http://www.flickr.com/photos/torek/3788181603/

"the **peace dividend** *of the* **smartphone wars***"*

- Chris Anderson

- Michael Scott

State of Hobby Robotics

Growing rapidly; lots of good magazines; costs have fallen; getting really approachable

State of
Hobby Robotics

https://github.com/esl/erlang_ale - GPIO Library for RaspPi

Client library landscape isn't fully mature. We had to build libraries for both the Sphero and the AR Drone

Why is Elixir a great choice?

Robotics is concurrent; Easy DSLs; Metaprogramming; it's just really really fun

Where to start?

Get easy wins early

Sphero: ~$130

ARDrone: ~$150-250

Sphero

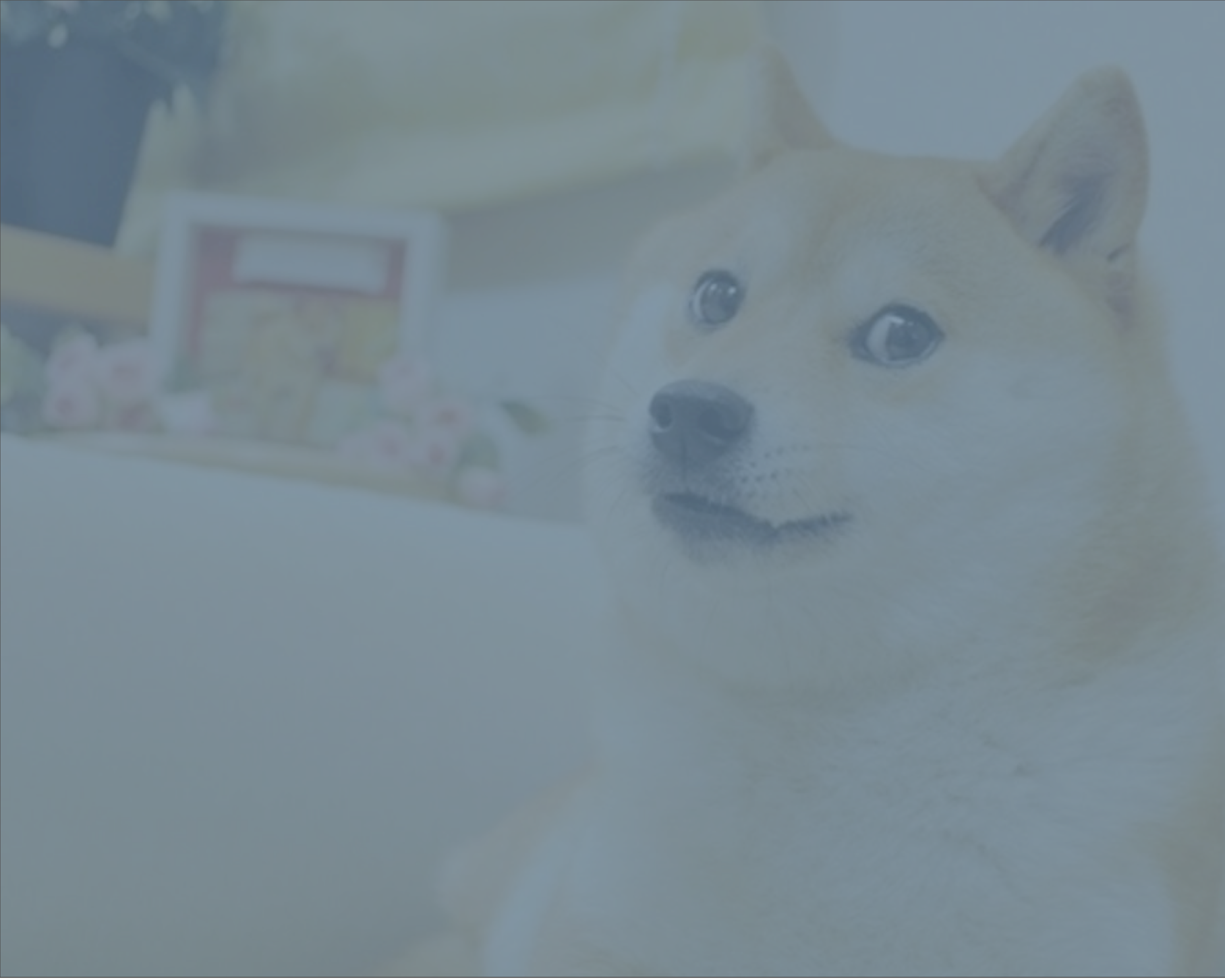
http://www.gosphero.com/

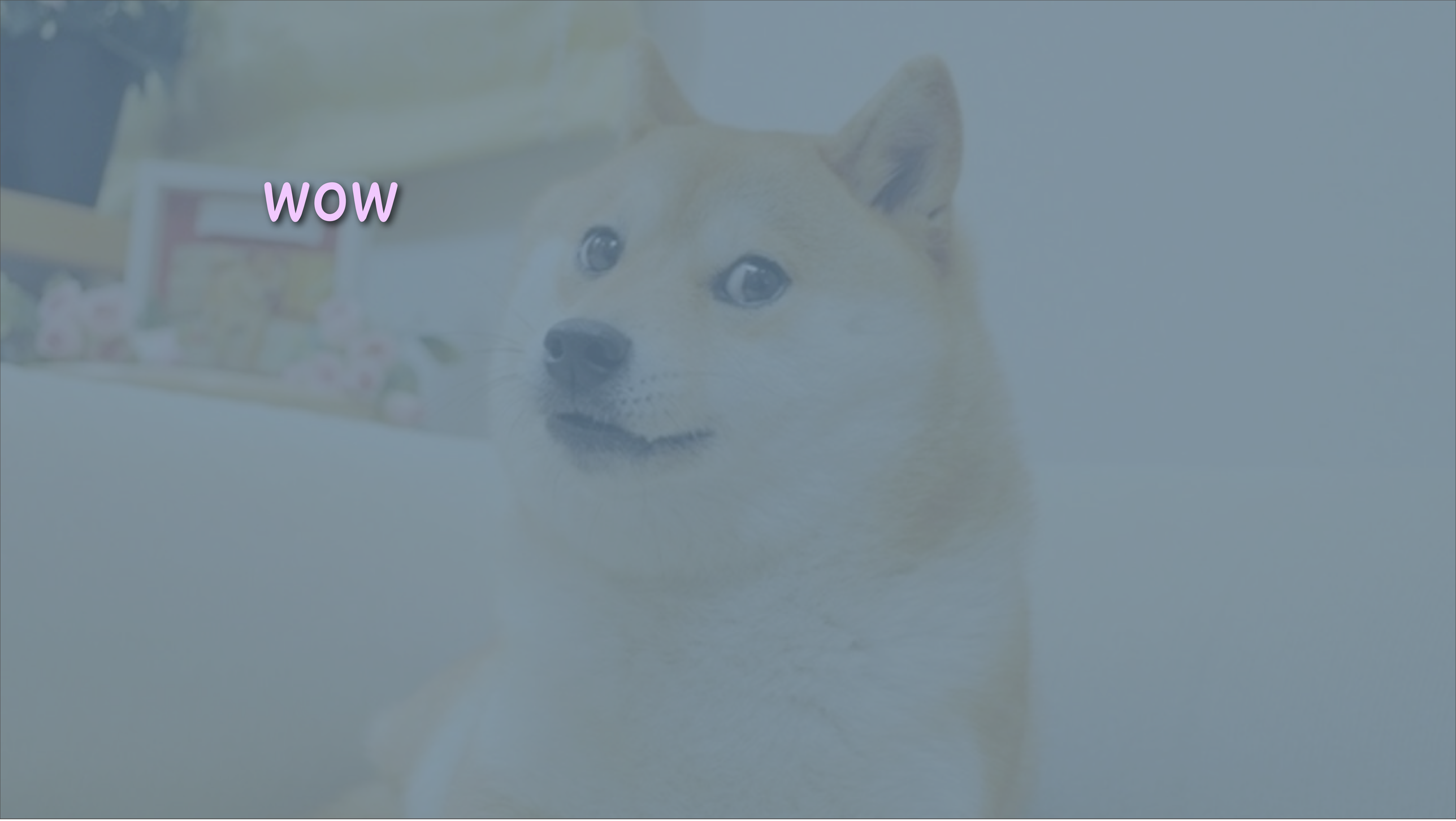
Bluetooth enabled ball; drives itself around; best dog toy ever; waterproof

```elixir
defmodule Examples.Square do
  alias Sphero.Client, as: C
  def roll device do
    {:ok, s} = C.start device
    :timer.sleep 4000
    Enum.map 1..5, &roll_square/1
    C.stop s
    Process.exit s, :kill
  end
  def roll_square s do
    roll_and_sleep s, 0
    roll_and_sleep s, 90
    roll_and_sleep s, 180
    roll_and_sleep s, 270
  end
  def roll_and_sleep s, angle do
    C.roll s, 80, angle
    :timer.sleep 1000
  end
end

Examples.Square.roll "/dev/rfcomm0"
```

```elixir
defmodule Examples.Square do
  alias Sphero.Client, as: C
  def roll device do
    {:ok, s} = C.start device
    :timer.sleep 4000
    Enum.map 1..5, &roll_square/1
    C.stop s
    Process.exit s, :kill
  end
  def roll_square s do
    roll_and_sleep s, 0
    roll_and_sleep s, 90
    roll_and_sleep s, 180
    roll_and_sleep s, 270
  end
  def roll_and_sleep s, angle do
    C.roll s, 80, angle
    :timer.sleep 1000
  end
end

Examples.Square.roll "/dev/rfcomm0"
```

```elixir
defmodule Examples.Square do
  alias Sphero.Client, as: C
  def roll device do
    {:ok, s} = C.start device
    :timer.sleep 4000
    Enum.map 1..5, &roll_square/1
    C.stop s
    Process.exit s, :kill
  end
  def roll_square s do
    roll_and_sleep s, 0
    roll_and_sleep s, 90
    roll_and_sleep s, 180
    roll_and_sleep s, 270
  end
  def roll_and_sleep s, angle do
    C.roll s, 80, angle
    :timer.sleep 1000
  end
end

Examples.Square.roll "/dev/rfcomm0"
```




wow


wow
very roll


wow
very roll
such distribute


wow
very roll
such distribute

Elixir Sphero Library
code spelunking

https://github.com/knewter/sphero

```elixir
defrecord Sphero.Request, seq: nil, data: "", did: <<0>>, cid: <<0>> do
  use Bitwise, only_operators: true

  def to_string(request) do
    packet_header(request) <> packet_body(request) <> checksum(request)
  end

  defp checksum(request) do
    csum = packet_header(request) <> packet_body(request)
             |> :binary.bin_to_list
             |> Enum.drop(2)
             |> sum
             |> rem(256)
    csumval = ~~~csum &&& 255
    <<csumval>>
  end

  defp header(request) do
    sop1 <>
    sop2 <>
    request.did <>
    request.cid <>
    <<request.seq>> <>
    :erlang.list_to_binary([dlen(request.data)])
  end
```

```elixir
defrecord Sphero.Client.State, device: nil, seq: nil
defmodule Sphero.Client do
  use ExActor.GenServer

  definit device do
    device = :serial.start([speed: 115200, open: bitstring_to_list(device)])
    initial_state(Sphero.Client.State.new(device: device, seq: 0))
  end

  defcall roll(speed, heading), state: state do
    do_request(Sphero.Command.Roll.new(seq: state.seq, speed: speed, heading: heading, delay: 1),
  end

  defp do_request(request, state) do
    request_bytes = Sphero.Request.to_string(request)
    send(state.device, {:send, request_bytes})
    # receive response
    _response = receive do
      {:data, data} -> IO.inspect data
    after
      1 -> :timeout
    end
    # update the seq
    state = state.seq(state.seq + 1)
    set_and_reply(state, :ok)
  end
```

Parrot ARDrone

http://ardrone2.parrot.com

Acts as its own wifi AP; Connect to it over wifi, it's just a device on the network; send UDP packets to it; stream video from 2 cameras.

```elixir
defmodule Examples.Basic do
  alias Exdrone.Drone, as: D

  def start do
    connection = Exdrone.Connection[host: {192,168,1,1}, port: 5556]
    {:ok, drone} = D.start(connection)
    drone |> D.take_off
    :timer.sleep(2000)
    drone |> D.hover
    :timer.sleep(1000)
    drone |> D.forward(0.1)
    :timer.sleep(2000)
    drone |> D.land
    Process.exit(drone, :kill)
  end
end
```

~
~
~
~
~

Elixir Exdrone Library
code spelunking

https://github.com/knewter/exdrone

```elixir
defmodule Exdrone.Drone do
  use ExActor.GenServer
  alias Exdrone.UdpSender
  alias Exdrone.AtCommander
  alias Exdrone.Controller

  defrecord State, controller: nil, seq: 1

  definit(connection // Exdrone.Connection[host: {192,168,1,1}, port: "5556"]) do
    sender             = UdpSender.start(connection)
    {:ok, commander}   = AtCommander.start(sender)
    {:ok, controller} = Controller.start(commander)

    initial_state(State[controller: controller])
  end

  defcall take_off, state: state do
    Controller.take_off(state.controller)
    set_and_reply(state, self)
  end
  # ... land, forward, hover, right, etc.
```

```elixir
defrecord Exdrone.UdpSender, [:connection, :socket] do
  def start(connection) do
    {:ok, socket} = :gen_udp.open(0, [:binary])
    Exdrone.UdpSender.new(connection: connection, socket: socket)
  end

  def send_packet(udp_sender, packet) do
    connection = udp_sender.connection
    :gen_udp.send(udp_sender.socket, connection.host, connection.port, packet)
  end
end
```

```elixir
defmodule Exdrone.AtCommander do
  use ExActor.GenServer
  alias Exdrone.AtCommander.State
  alias Exdrone.UdpSender

  defrecord ServerState, commander_state: nil, sender: nil, timer: nil

  definit(sender) do
    {:ok, timer} = :timer.apply_interval(30, Exdrone.AtCommander, :tick, [self])
    initial_state(ServerState[commander_state: State.new, sender: sender, timer:
  end

  defcall tick, state: state do
    commander_state = state.commander_state |> State.build_tick
    message = commander_state |> State.build_message
    state.sender |> UdpSender.send_packet(message)
    commander_state = commander_state.buffer("")
    state = state.commander_state(commander_state)
    set_and_reply(state, self)
  end

```

```elixir
defmodule Exdrone.Controller do
  alias Exdrone.AtCommander
  use ExActor.GenServer
  #...
  definit(at_commander) do
    state = State[at_commander: at_commander]
    update_ref(state)
    calibrate(state)
    initial_state(state)
  end

  defcall take_off, state: state do
    state = state.flying(true)
    state = state.emergency(false)
    state = update_ref(state)
    set_and_reply(state, self)
  end

  def update_ref(state) do
    n = ref_base
    if state.flying, do: n = n |> bor(ref_fly_bit)
    if state.emergency, do: n = n |> bor(ref_emergency_bit)
    state.at_commander(state.at_commander |> AtCommander.ref(n))
  end

  defcall land, state: state do
```

Rolling your own

What if you don't want a ball or a quadcopter?

What if you want to be master of your own fate?

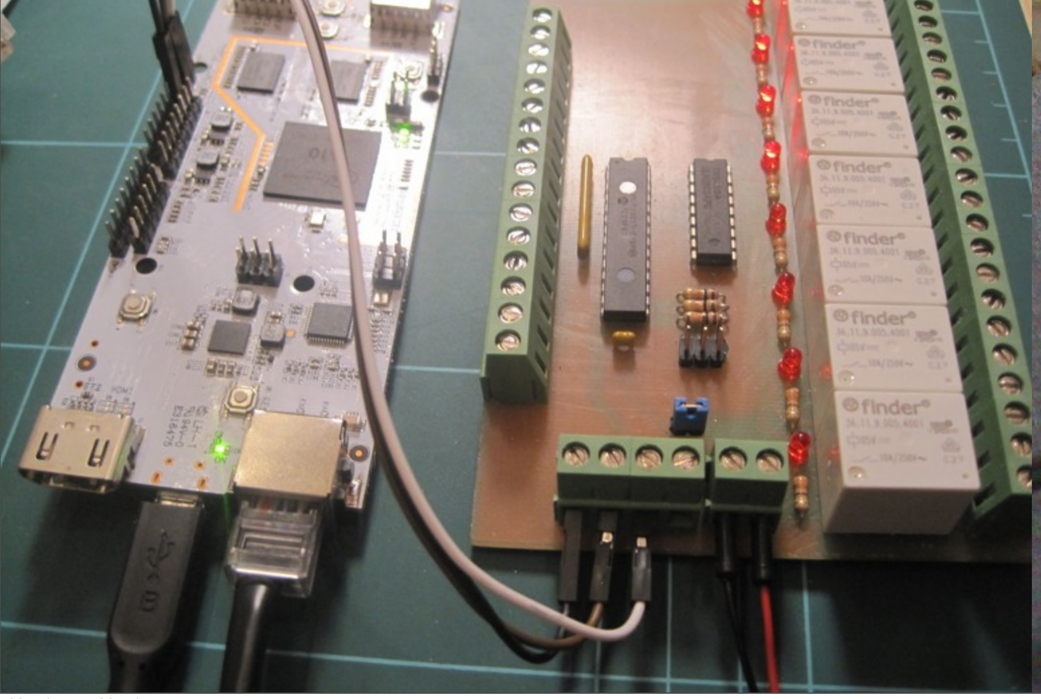
Microcontrollers

A microcontroller is **a small computer on a single IC** containing a processor, memory, and programmable I/O.

Microcontrollers!

Microcontrollers!

Arduino



Microcontrollers!

Arduino

Beaglebone Black

Microcontrollers!

Arduino

Beaglebone Black

PCDuino

Microcontrollers!

Arduino

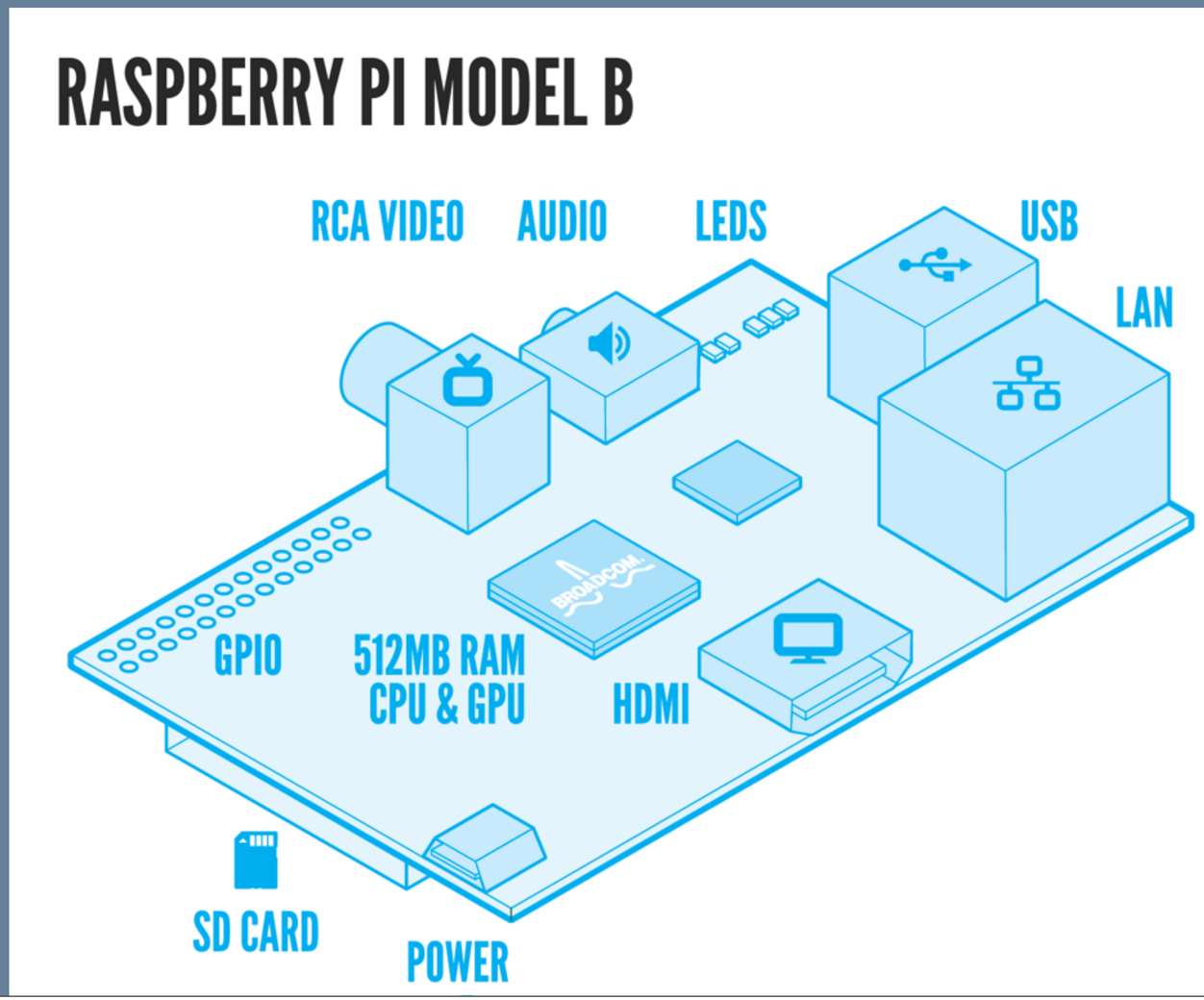
Beaglebone Black

PCDuino

Raspberry Pi

Raspberry Pi

The Raspberry Pi is a **credit-card sized computer** that plugs into your TV and a keyboard. It's a capable little PC which can be used for many of the things that your desktop PC does, like spreadsheets, word-processing and games. It also plays high-definition video.



RaspberryPi

Plug in the SD card; plug it all up (power last); answer a dozen setup questions.

RaspberryPi:
Running **Erlang** and Elixir

```
git clone git@github.com:erlang/otp; cd otp;
./configure && make && make install
```

wait 3 hours
(one Lord of the Ring)

RaspberryPi:
Running Erlang and **Elixir**

```
git clone git@github.com:elixir-lang/elixir;
cd elixir; make
```

wait 20 minutes

RaspberryPi GPIO and Tips

Limited hardware PWM by default (there's a way around this); use a 1+ amp power supply

RaspberryPi
Tips (cont)

Get a Pi Cobbler; Cirago wifi/bluetooth dongle ($11); buy a nice case or print one; use a class 10 SD card; camera module for $25

A80 Optimus Board

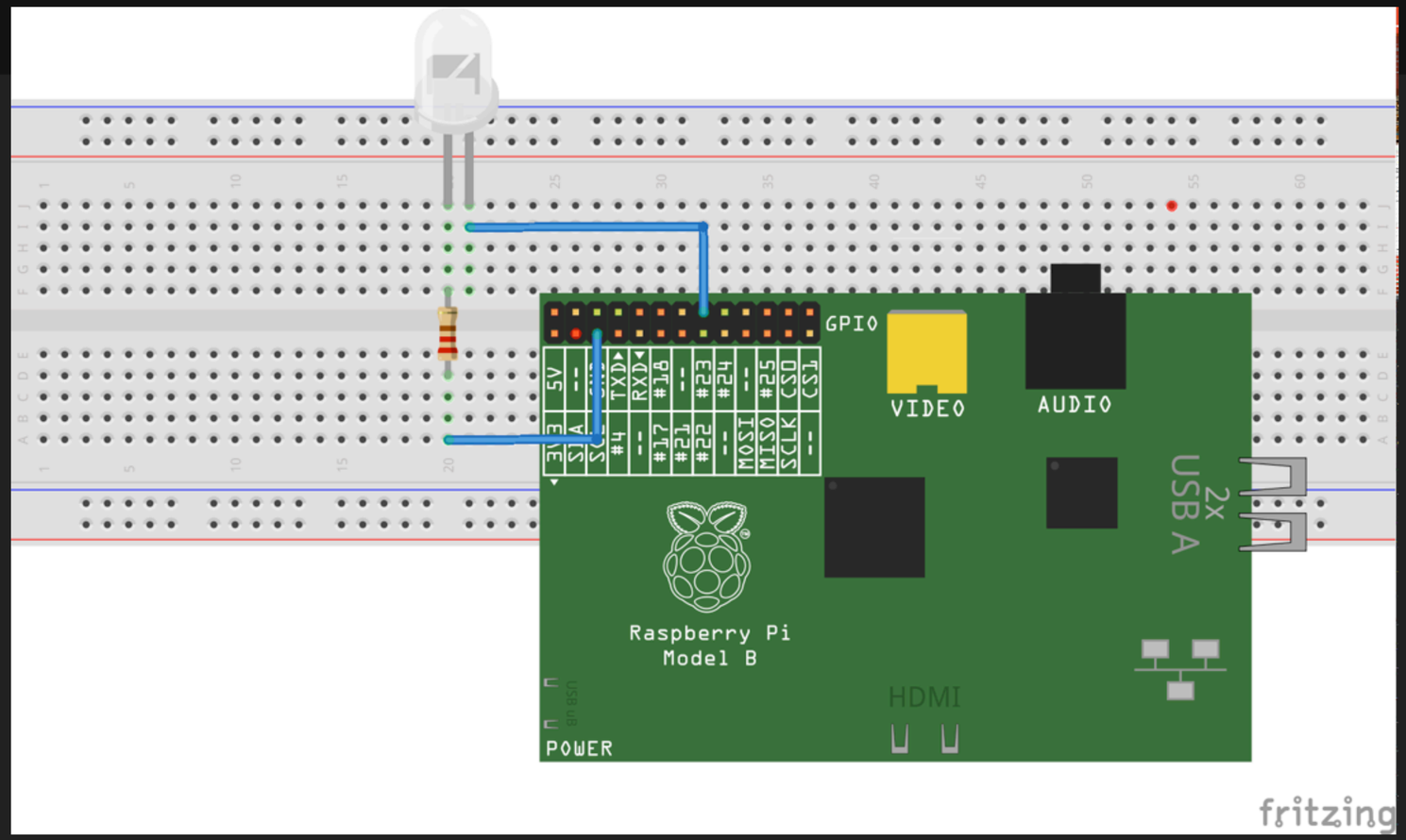
Small single board computer; 8-core processor, USB, GPIO, etc.

GPIO Examples

erlang_ale https://github.com/esl/erlang_ale

Event driven Raspberry Pi GPIO programming in Elixir

```elixir
defmodule PinFun do
  def blink do
    :gpio_sup.start_link([{25, :output}])
    :gpio.write(25, 1)
    :timer.sleep(1000)
    :gpio.write(25, 0)
  end
end
```



Control schemes

Serial Ports; Bluetooth; USB; GPIO; Erlang Distribution

Part the Building something neat

(This slide intentionally left dumb)

What's next?

We can blink an LED; next quick win? Our goal is to demystify, rather than show detailed code

Give me control!

How about some more direct control?

Android -> JInterface -> Elixir -> RasPi -> Blinkenlight

JInterface on Android Hurdles

- Have to be running epmd on the host machine
- Android sandboxing doesn't let you install a 'generic' erlang that you can use from other apps (unless you're rooted)
- So you have to ship Erlang with your android app

Installing Erlang
on android

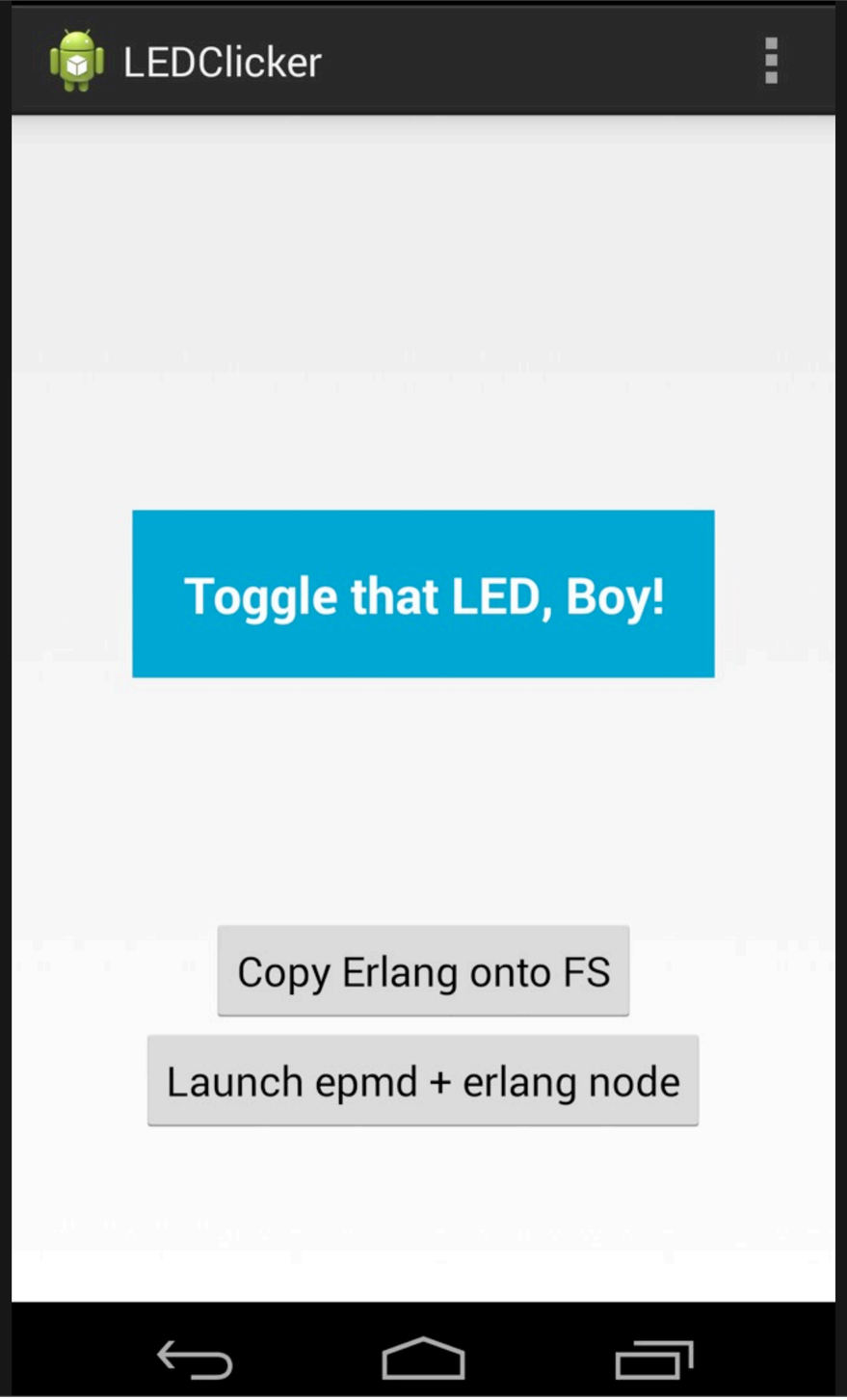
So now android?

Eww, Java; not the worst thing in the world; tiny code.

```
copyButton.setOnClickListener(new View.OnClickListener() {
    public void onClick(View v) {
        copyErlangOntoFs();
        makeExecutable("/erlang/bin/epmd");
        makeExecutable("/erlang/bin/erl");
    }
});

launchErlangButton.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        launchEpmd();
        launchErlangNode();
        mIsReady = true;
    }
});
```

```java
public void copyErlangOntoFs() {
    Log.d(TAG, "copyErlangOntoFs start");

    InputStream erlangZipFileInputStream = null;
    try {
        erlangZipFileInputStream = getActivity().getApplicationContext().getAssets().open("erlang_R16B.zip");
    } catch (IOException e) {
        e.printStackTrace();
    }
    Log.d(TAG, erlangZipFileInputStream.toString());
    Log.d(TAG, MainActivity.context.getFilesDir().getPath());
    Decompress unzipper = new Decompress(erlangZipFileInputStream, MainActivity.context.getFilesDir().getPath() + "/");
    unzipper.unzip();

    Log.d(TAG, "copyErlangOntoFs done");
}
```

```
D/RGBLedSetterDecompress: Unzipping erlang/lib/kernel-2.16.1/ebin/erl_boot_server.beam into /data/data/com.isotopel
D/RGBLedSetterDecompress: Unzipping erlang/lib/kernel-2.16.1/ebin/user_drv.beam into /data/data/com.isotopell.rccon
D/RGBLedSetterDecompress: Unzipping erlang/lib/kernel-2.16.1/ebin/file.beam into /data/data/com.isotopell.rccontrol
D/RGBLedSetterDecompress: Unzipping erlang/lib/kernel-2.16.1/ebin/dist_ac.beam into /data/data/com.isotopell.rccont
D/RGBLedSetterDecompress: Unzipping erlang/lib/kernel-2.16.1/ebin/inet_dns.beam into /data/data/com.isotopell.rccon
D/RGBLedSetterDecompress: Unzipping erlang/lib/kernel-2.16.1/ebin/net.beam into /data/data/com.isotopell.rccontroll
D/RGBLedSetterDecompress: Unzipping erlang/lib/kernel-2.16.1/ebin/inet_hosts.beam into /data/data/com.isotopell.rcc
D/RGBLedSetterDecompress: Unzipping erlang/lib/kernel-2.16.1/ebin/wrap_log_reader.beam into /data/data/com.isotopel
D/RGBLedSetterDecompress: Unzipping erlang/lib/kernel-2.16.1/ebin/inet_tcp.beam into /data/data/com.isotopell.rccon
D/RGBLedSetterDecompress: Unzipping erlang/lib/kernel-2.16.1/ebin/heart.beam into /data/data/com.isotopell.rccontro
D/RGBLedSetterDecompress: Unzipping erlang/lib/kernel-2.16.1/ebin/global_group.beam into /data/data/com.isotopell.r
D/RGBLedSetterDecompress: Unzipping erlang/lib/kernel-2.16.1/ebin/disk_log_1.beam into /data/data/com.isotopell.rcc
D/RGBLedSetterDecompress: Unzipping erlang/lib/kernel-2.16.1/ebin/ram_file.beam into /data/data/com.isotopell.rccon
D/RGBLedSetterDecompress: Unzipping erlang/lib/kernel-2.16.1/ebin/inet_res.beam into /data/data/com.isotopell.rccon
```


LEDClicker
Toggle that LED, Boy!
Copy Erlang onto FS
Launch epmd + erlang node

```
201         }
202       } catch (IOException e) {
203         e.printStackTrace();
204       }
205     }
206   }
207
208   @Override
209   protected void onPostExecute(String result) {
210     //Log.e(TAG, "onPostExecute");
211   }
212
213   public void toggleLed(){
214     cast(new OtpErlangAtom("toggle"));
215   }
216
217   public void cast(OtpErlangObject message){
218     OtpErlangObject[] castMsg = new OtpErlangObject[2];
219     castMsg[0] = new OtpErlangAtom("$gen_cast");
220     castMsg[1] = message;
221
222     mbox.send("led", remoteNodeName, new OtpErlangTuple(castMsg));
223     Log.d(TAG, "cast completed");
224   }
225 }
226 }
```

NORMAL <ker/src/main/java/com/isotope11/ledclicker/MainActivity.java java utf-8[unix] 100% ㏑ 226: 1

```elixir
defmodule PinServer.Server do
  use ExActor.GenServer

  definit(pin) do
    :gpio_sup.start_link([{pin, :output}])
    initial_state({0, pin})
  end

  defcast toggle, state: {0, pin} do
    :gpio.write(pin, 1)
    new_state({1, pin})
  end
  defcast toggle, state: {1, pin} do
    :gpio.write(pin, 0)
    new_state({0, pin})
  end
end
```

~
~
~
~
~
~
~
~
~
~

```elixir
defmodule PinServer.Server do
  use ExActor.GenServer

  definit(pin) do
    :gpio_sup.start_link([{pin, :output}])
    initial_state({0, pin})
  end

  defcast toggle, state: {0, pin} do
    :gpio.write(pin, 1)
    new_state({1, pin})
  end
  defcast toggle, state: {1, pin} do
    :gpio.write(pin, 0)
    new_state({0, pin})
  end
end
```

PinServer

```sh
{:ok, s} = PinServer.Server.start(25)
:erlang.register(:led, s)
```

You can run it on a node thusly:

```
iex --name "server@192.168.1.10" --cookie test \
    -pa _build/dev/lib/pin_server/ebin/ \
    -pa _build/dev/lib/exactor/ebin/
```

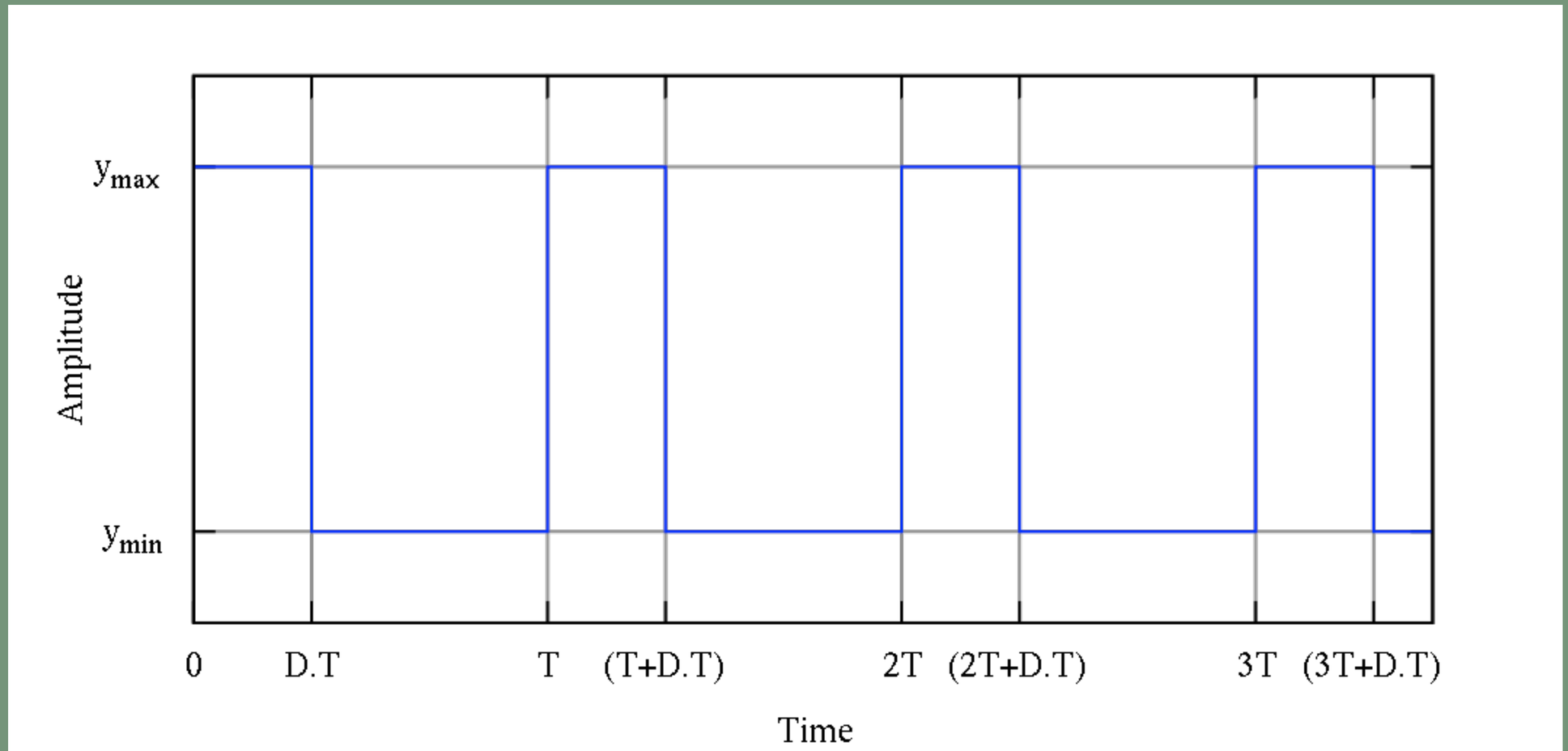
Then the Android app can talk to the server.

der blinkenlights

RGB LED?

- We need another quick win. What's next?
- RGB LED, controlled by a color picker on the android device, that changes color in realtime.
- Problem: RGB LED requires analog levels, but we only have digital with erlang_ale.

Pulse Width Modulation



Pulse Width Modulation

https://github.com/sarfata/pi-blaster/

Only one hardware PWM pin; pi-blaster gives you 8 PWM outputs.

Pulse Width Modulation

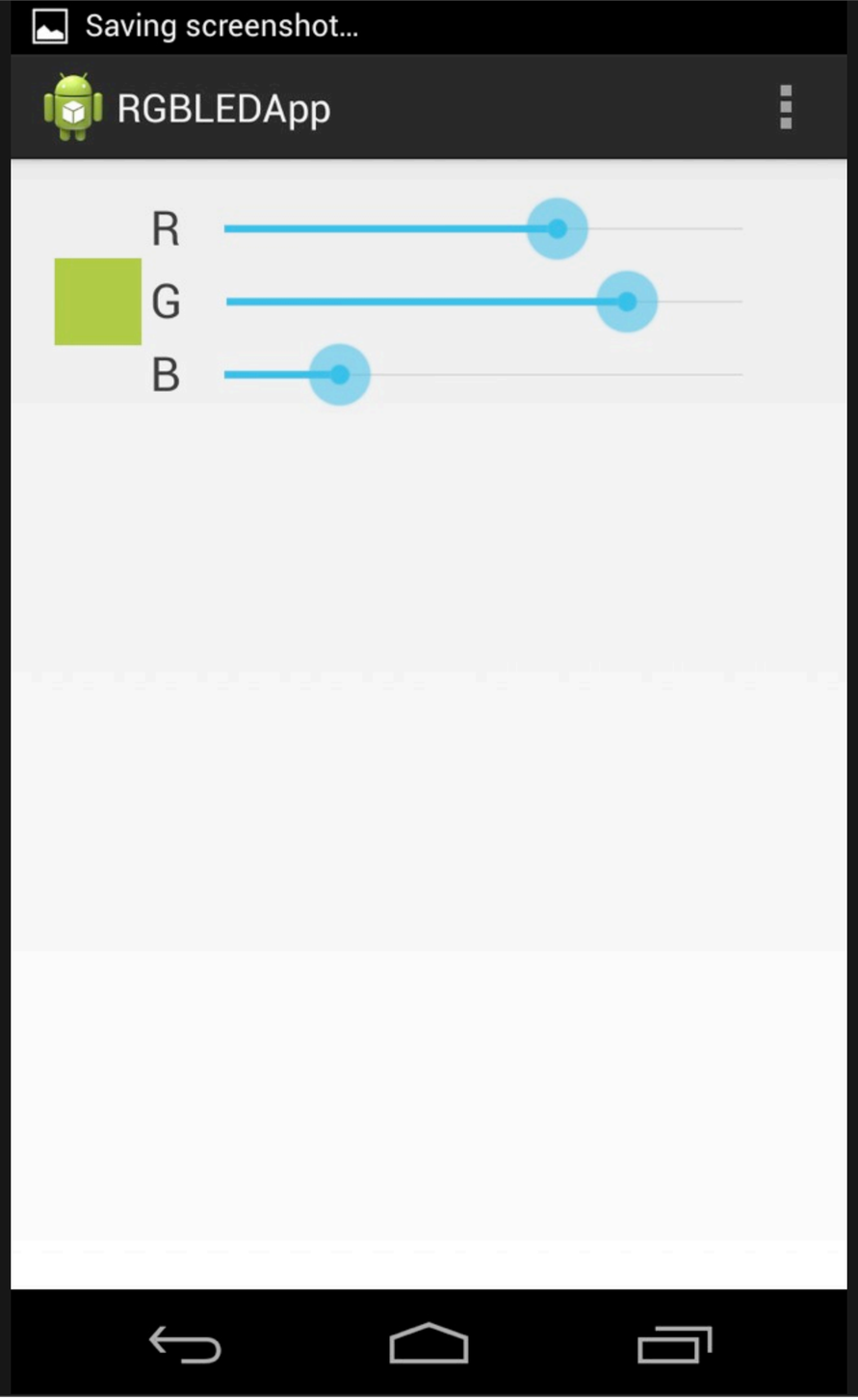
0% `echo "23=0" > /dev/pi-blaster`

100% `echo "23=1" > /dev/pi-blaster`

20% `echo "23=0.2" > /dev/pi-blaster`

```elixir
defmodule RgbLed.Led do
  defrecord Pin, number: nil, value: 0
  defrecord Component, [:red, :green, :blue]

  def init(red, green, blue) do
    Component[red: Pin[number: red],
              green: Pin[number: green],
              blue: Pin[number: blue]]
  end

  def pi_blast(component) do
    component |> pi_blast(:red)
    component |> pi_blast(:green)
    component |> pi_blast(:blue)
  end

  def set_value(component, color, value) do
    pin = component |> get_pin(color)
    new_pin = pin.value(value)
    case color do
      :red   -> component.red(new_pin)
      :green -> component.green(new_pin)
      :blue  -> component.blue(new_pin)
    end
  end

  def get_pin(component, color) do
    case color do
      :red   -> component.red
      :green -> component.green
      :blue  -> component.blue
    end
  end

  def get_value(component, color) do
    get_pin(component, color).value
  end

  def inverted_value(value), do: 1 - value

  def pi_blast(component, color) do
    pin = get_pin(component, color)
    PiBlaster.set_pin(pin.number, inverted_va
  end
end
```

```elixir
defmodule RgbLed.Server do
  use ExActor
  alias RgbLed.Led

  def init([red, green, blue]) do
    Led.init(red, green, blue) |> initial_state
  end

  defcast update(values), state: state do
    do_update(values, state) |> new_state
  end
  defcast red(value), state: state do
    IO.puts "red!, #{value}"
    state |> Led.set_value(:red, value) |> new_state
  end
  defcast green(value), state: state do
    IO.puts "green, #{value}"
    state |> Led.set_value(:green, value) |> new_state
  end
  defcast blue(value), state: state do
    IO.puts "blue, #{value}"
    state |> Led.set_value(:blue, value) |> new_state
  end
  defcast blast, state: state do
    IO.puts "blast"
    state |> Led.pi_blast
  end

  defp do_update([], state), do: state
  defp do_update([h|t], state) do
    new_state = do_update(h, state)
    do_update(t, new_state)
  end
  defp do_update({color, value}, state) do
    state |> Led.set_value(color, value)
  end
end
```


Saving screenshot...
RGBLEDApp
R
G
B

```
     */
    public class PlaceholderFragment extends Fragment {

        public PlaceholderFragment() {
        }

        @TargetApi(Build.VERSION_CODES.HONEYCOMB)
        @Override
        public View onCreateView(LayoutInflater inflater, ViewGroup container,
                Bundle savedInstanceState) {
            View rootView = inflater.inflate(R.layout.fragment_main, container, false);

            ColorMixer colors = (ColorMixer) rootView.findViewById(R.id.mixer);

            TimerTask task = new TimerTask() {
                public void run(){
                    if(mIsReady){
                        RGBLedSetter task = new RGBLedSetter();
                        task.execute();
                    }
                }
            };
            Timer timer = new Timer();
            timer.schedule(task, 0, 100);

            colors.setOnColorChangedListener(new ColorMixer.OnColorChangedListener() {
```

```
216     public class RGBLedSetter extends AsyncTask<Object, Void, String> {
217         final String remoteNodeName = "server@192.168.1.10";
218
219         @Override
220         protected String doInBackground(Object... arg0) {
221             prepareNode();
222             updateLed();
223             return "whatevs...";
224         }
225
226         public void prepareNode(){
227             if(self == null){
228                 try {
229                 self = new OtpNode("mynode", COOKIE);
230                 mbox = self.createMbox("rgbcolorpicker");
231                 if (self.ping(remoteNodeName, 2000)) {
232                     System.out.println("remote is up");
233                 } else {
234                     System.out.println("remote is not up");
235                     return;
236                 }
237             } catch (IOException e) {
238                 e.printStackTrace();
239             }
240         }
241     }
```

```
216     public class RGBLedSetter extends AsyncTask<Object, Void, String> {
217         final String remoteNodeName = "server@192.168.1.10";
218 
219         @Override
220         protected String doInBackground(Object... arg0) {
221             prepareNode();
222             updateLed();
223             return "whatevs...";
224         }
225 
226         public void prepareNode(){
227             if(self == null){
228                 try {
229                     self = new OtpNode("mynode", COOKIE);
230                     mbox = self.createMbox("rgbcolorpicker");
231                     if (self.ping(remoteNodeName, 2000)) {
232                         System.out.println("remote is up");
233                     } else {
234                         System.out.println("remote is not up");
235                         return;
236                     }
237                 } catch (IOException e) {
238                     e.printStackTrace();
239                 }
240             }
241         }
```

```
247
248         public void updateLed(){
249             cast(valueUpdate("red", red));
250             cast(valueUpdate("green", green));
251             cast(valueUpdate("blue", blue));
252             cast(new OtpErlangAtom("blast"));
253         }
254
255         public OtpErlangTuple valueUpdate(String color, Float value){
256             OtpErlangObject[] message = new OtpErlangObject[2];
257             message[0] = new OtpErlangAtom(color);
258             message[1] = new OtpErlangFloat(value);
259
260             return new OtpErlangTuple(message);
261         }
262
263         public void cast(OtpErlangObject message){
264             OtpErlangObject[] castMsg = new OtpErlangObject[2];
265             castMsg[0] = new OtpErlangAtom("$gen_cast");
266             castMsg[1] = message;
267
268             mbox.send("rgbled", remoteNodeName, new OtpErlangTuple(castMsg));
269             Log.d(TAG, "cast completed");
270         }
271     }
```

```
247
248         public void updateLed(){
249             cast(valueUpdate("red", red));
250             cast(valueUpdate("green", green));
251             cast(valueUpdate("blue", blue));
252             cast(new OtpErlangAtom("blast"));
253         }
254
255         public OtpErlangTuple valueUpdate(String color, Float value){
256             OtpErlangObject[] message = new OtpErlangObject[2];
257             message[0] = new OtpErlangAtom(color);
258             message[1] = new OtpErlangFloat(value);
259
260             return new OtpErlangTuple(message);
261         }
262
263         public void cast(OtpErlangObject message){
264             OtpErlangObject[] castMsg = new OtpErlangObject[2];
265             castMsg[0] = new OtpErlangAtom("$gen_cast");
266             castMsg[1] = message;
267
268             mbox.send("rgbled", remoteNodeName, new OtpErlangTuple(castMsg));
269             Log.d(TAG, "cast completed");
270         }
271     }
```

```
247
248         public void updateLed(){
249             cast(valueUpdate("red", red));
250             cast(valueUpdate("green", green));
251             cast(valueUpdate("blue", blue));
252             cast(new OtpErlangAtom("blast"));
253         }
254
255         public OtpErlangTuple valueUpdate(String color, Float value){
256             OtpErlangObject[] message = new OtpErlangObject[2];
257             message[0] = new OtpErlangAtom(color);
258             message[1] = new OtpErlangFloat(value);
259
260             return new OtpErlangTuple(message);
261         }
262
263         public void cast(OtpErlangObject message){
264             OtpErlangObject[] castMsg = new OtpErlangObject[2];
265             castMsg[0] = new OtpErlangAtom("$gen_cast");
266             castMsg[1] = message;
267
268             mbox.send("rgbled", remoteNodeName, new OtpErlangTuple(castMsg));
269             Log.d(TAG, "cast completed");
270         }
271     }
```


AFFORDABLE
TOWING
724-2241

So what's next?

Finish up with something fun: spend $2 on a toy at the thrift store. **Make it amazing.**

Take a Sabertooth 2x12RC ($65)

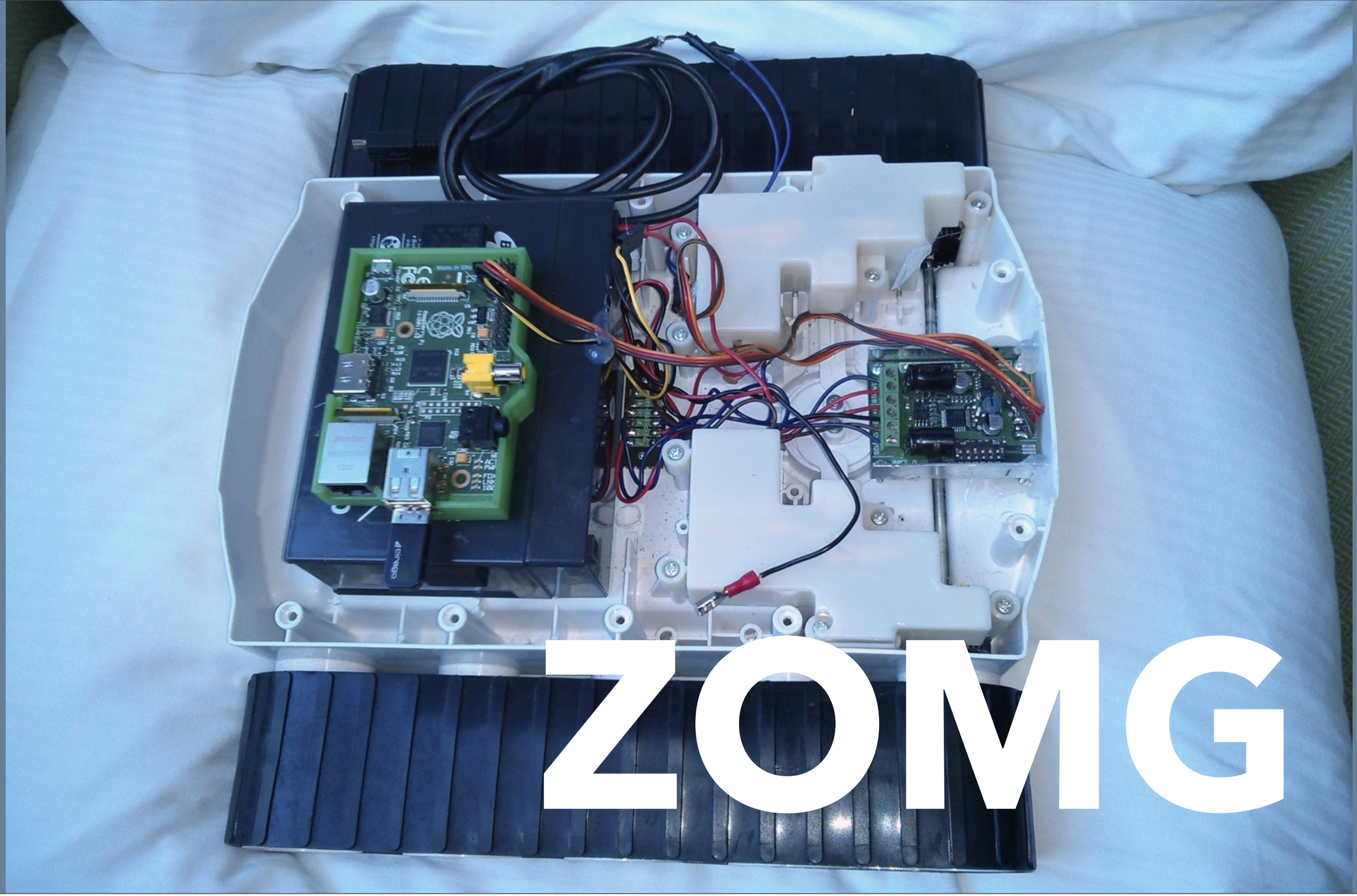
2x12RC ($65), add a Raspberry Pi, add a 12V

a Raspberry Pi, add a 12V battery, add a Ci

2V battery, add a Cirago WiFi/BT dongle mu

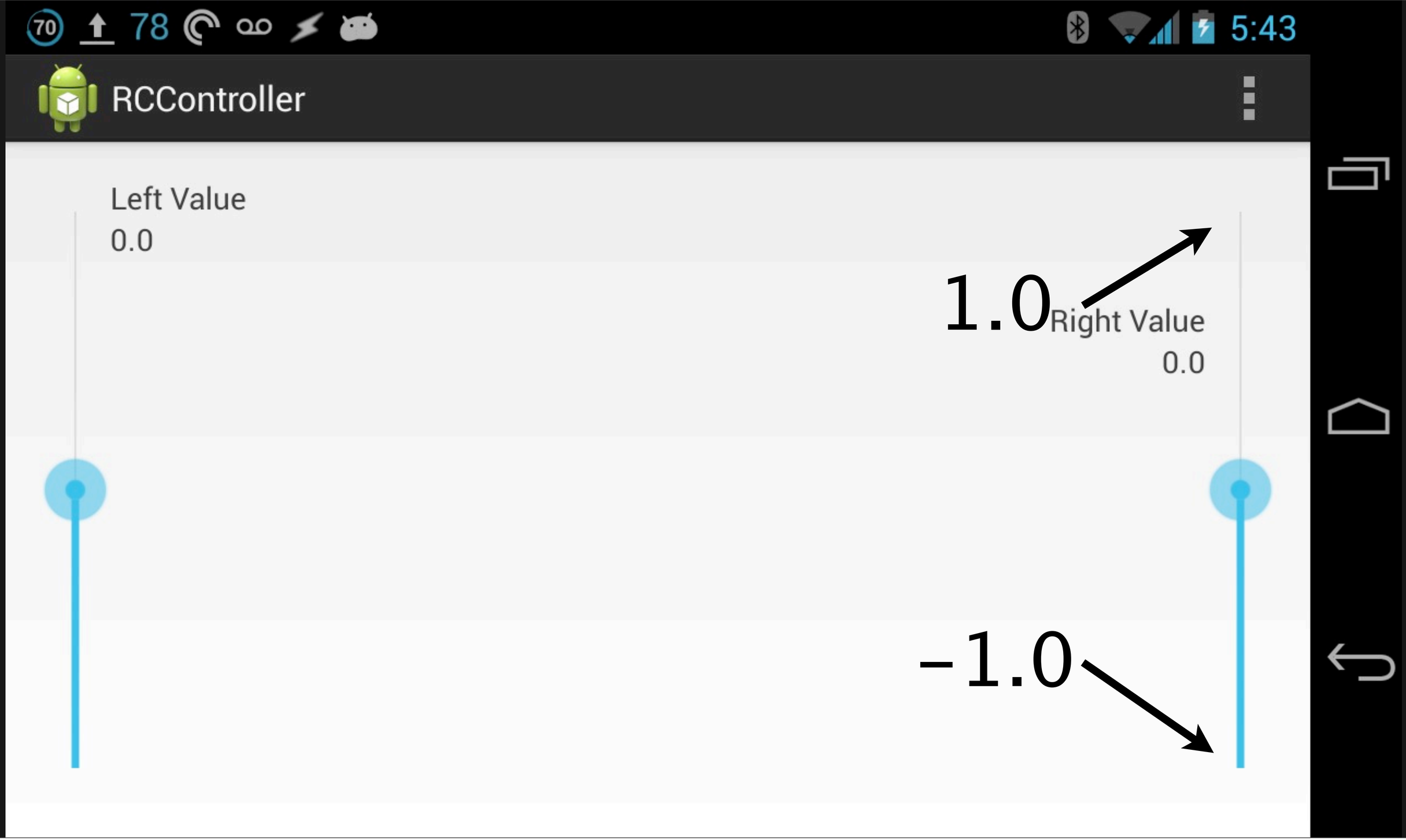
BT dongle, mutilate a USB micro cable, and y

3 micro cable, and you end up with this.


ZOMG

PWM and the Sabertooth

2000ms **100% forward**
1500ms **idle**
1000ms **100% reverse**


5:43
RCController
Left Value
0.0
1.0
Right Value
0.0
−1.0

```
315         public void updateTank(){
316             castUpdate((float) mLeft, (float) mRight);
317             castBlast();
318         }
319
320         public void castUpdate(Float left, Float right){
321             OtpErlangObject[] message = new OtpErlangObject[3];
322             message[0] = new OtpErlangAtom("update");
323             message[1] = new OtpErlangFloat(left);
324             message[2] = new OtpErlangFloat(right);
325
326             cast(new OtpErlangTuple(message));
327         }
328
329         public void castBlast(){
330             cast(new OtpErlangAtom("blast"));
331         }
332
333         public void cast(OtpErlangObject message){
334             //Log.e(TAG, "updateLed");
335             OtpErlangObject[] castMsg = new OtpErlangObject[2];
336             castMsg[0] = new OtpErlangAtom("$gen_cast");
337             castMsg[1] = message;
338
339             mbox.send("raspi_tank", remoteNodeName, new OtpErlangTuple(castMsg));
340             Log.d(TAG, "cast completed");
```

```elixir
defmodule RaspiTank.Motor do
  defrecord Pin, number: nil, value: nil

  def init(pin), do: Pin[number: pin]

  def set_speed(value, pin) do
    mapped_value_for(value) |> pin.value
  end

  def pi_blast(Pin[number: number, value: value]) do
    PiBlaster.set_pin(number, value)
  end

  defp mapped_value_for(value) when value >= -1 and value <= 1 do
    idle + (value * 500) / max_pulse_width
  end

  defp idle, do: 1_500 / max_pulse_width
  defp max_pulse_width, do: 10_000
end
```

~
~
~
~
~
~

```elixir
defmodule RaspiTank.Motor do
  defrecord Pin, number: nil, value: nil

  def init(pin), do: Pin[number: pin]

  def set_speed(value, pin) do
    mapped_value_for(value) |> pin.value
  end

  def pi_blast(Pin[number: number, value: value]) do
    PiBlaster.set_pin(number, value)
  end

  defp mapped_value_for(value) when value >= -1 and value <= 1 do
    idle + (value * 500) / max_pulse_width
  end

  defp idle, do: 1_500 / max_pulse_width
  defp max_pulse_width, do: 10_000
end
```

~
~
~
~
~
~

```elixir
defmodule RaspiTank.Motor do
  defrecord Pin, number: nil, value: nil

  def init(pin), do: Pin[number: pin]

  def set_speed(value, pin) do
    mapped_value_for(value) |> pin.value
  end

  def pi_blast(Pin[number: number, value: value]) do
    PiBlaster.set_pin(number, value)
  end

  defp mapped_value_for(value) when value >= -1 and value <= 1 do
    idle + (value * 500) / max_pulse_width
  end

  defp idle, do: 1_500 / max_pulse_width
  defp max_pulse_width, do: 10_000
end
```

~
~
~
~
~
~

```elixir
defmodule RaspiTank.Tank do
  alias RaspiTank.Motor

  def init(left, right) do
    left_motor = Motor.init(left)
    right_motor = Motor.init(right)
    {:tank, left_motor, right_motor}
  end

  def set_speed(left, right, {:tank, left_motor, right_motor}) do
    {:tank,
       Motor.set_speed(left, left_motor),
       Motor.set_speed(right, right_motor)}
  end

  def pi_blast({:tank, left_motor, right_motor}) do
    Motor.pi_blast(left_motor)
    Motor.pi_blast(right_motor)
  end
end
```

```
~
~
~
~
~
~
~
```

```
NORMAL  master  lib/raspi_tank/tank.ex                elixir  utf-8[unix]  5% ㏑  1
```

```elixir
defmodule RaspiTank.Server do
  @moduledoc """
  To start a Tank Server, just do:

      iex> {:ok, pid} = RaspiTank.Server.start(23, 24)
      iex> pid |> RaspiTank.Server.update(left_speed, right_speed)
      iex> pid |> RaspiTank.Server.blast

  """

  use ExActor.GenServer
  alias RaspiTank.Tank

  definit([left_pin, right_pin]) do
    Tank.init(left_pin, right_pin) |> initial_state
  end

  defcast update(left_speed, right_speed), state: state do
    IO.puts "update(#{left_speed}, #{right_speed})"
    Tank.set_speed(left_speed, right_speed, state) |> new_state
  end

  defcast blast, state: state do
    state |> Tank.pi_blast
    noreply
  end
```

```
# RaspiTank

In `examples/run_tank.exs` you'll find:

```sh
{:ok, s} = RaspiTank.Server.start([23, 24])
:erlang.register(:raspi_tank, s)
```

You can run it on a node thusly:

```
iex --name "server@192.168.1.10" --cookie test \
    -pa _build/dev/lib/raspi_tank/ebin/ \
    -pa _build/dev/lib/exactor/ebin/ \
    -r examples/run_tank.exs
```

Then the Android app can talk to the server.
~
~
~
~
~
~
~
~
```










Monday, 10 March 14

Thank you for coming

@knewter @robby_clements

www.elixirsips.com

isotope11.com

irc: freenode#isotope11

Check out our blog, or else.


/rclements/erlang_factory_robots