FLY LIKE AN EAGLE

BUILD A QUADCOPTER FROM SCRATCH

WHO IS THIS GUY?

I'M LOÏC

@loicvigneron

loic@spin42.com


SPIN42

THE IDEA

I WANTED TO FIND SOMETHING TO CHILL OUT

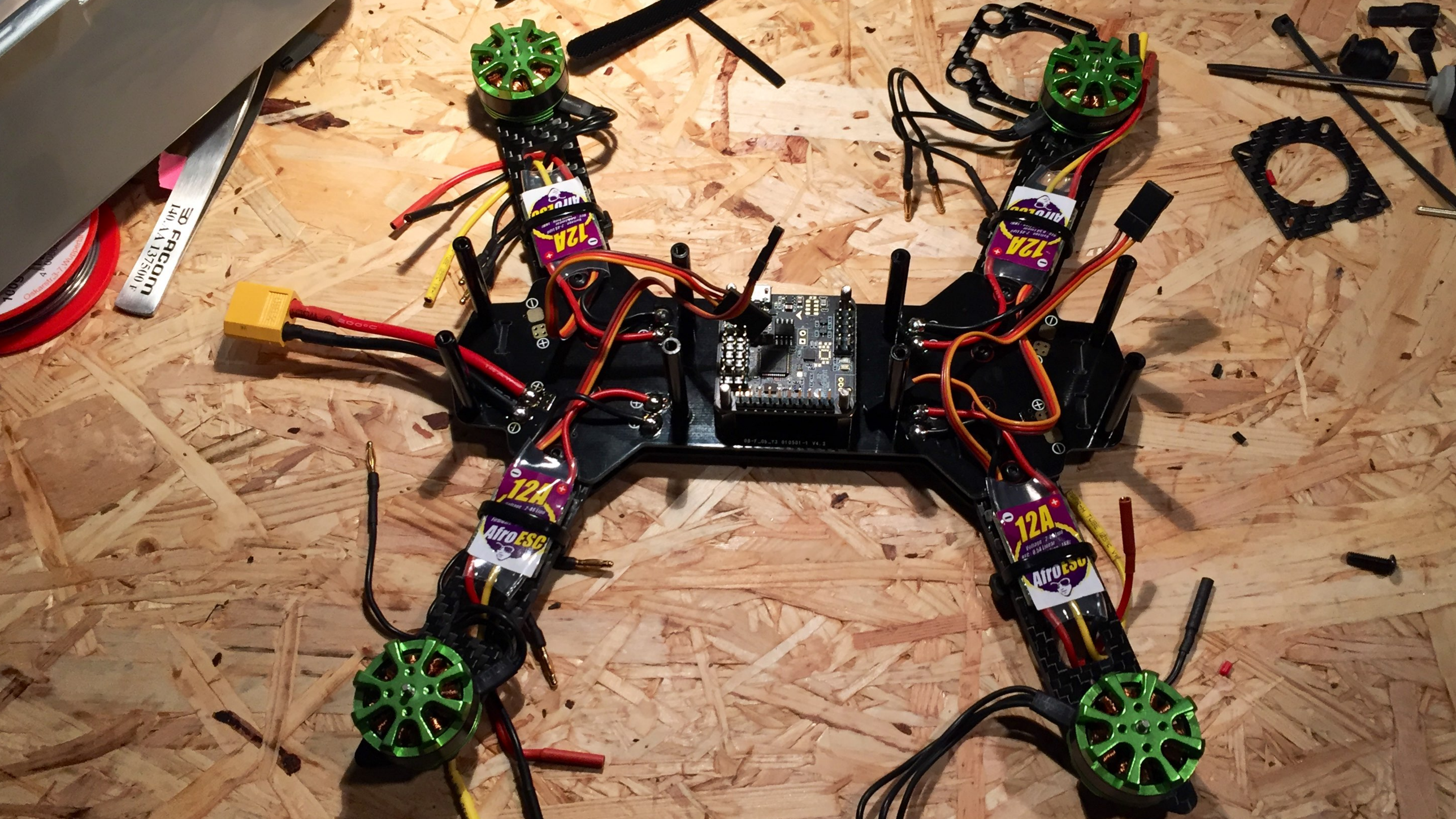
BIGGEST DRONE RACE

WORLD DRONE PRIX DUBAI

1st prize: $250,000

THE IDEA

THAT'S MY GAME!
LET'S BUY ONE


FACOM
12A
AfroESC

THE IDEA

I WANTED TO KNOW HOW IT WORKS.
LET'S BUILD ONE.

WHAT TO BUILD & FOR WHOM?

DON’T BUILD FOR /DEV/NULL

UNDERSTANDABLE

HUMAN FRIENDLY/ READABLE SOURCE CODE. WELL SEPARATED BUILDING BLOCKS

EXTENDABLE

EASY TO CONNECT HARDWARE, 3D PRINTED PIECES

DESIGN CHOICES

BUILDING BLOCKS

EVERYTHING TOGETHER

FURTHER WORK

DESIGN CHOICES

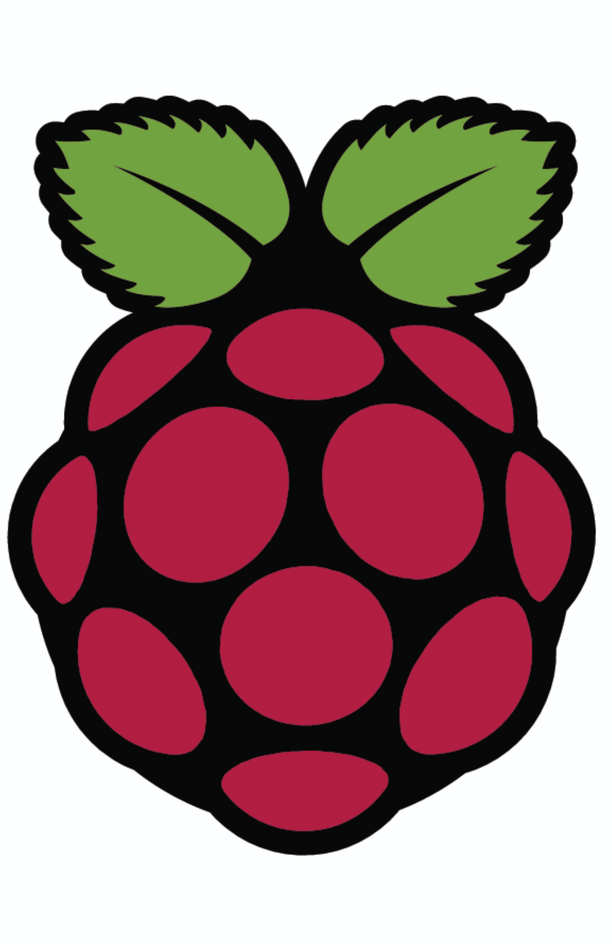
BUILDING BLOCKS

EVERYTHING TOGETHER

FURTHER WORK

Compute unit

Tools



Raspberry PI 3

Affordable

1GB memory

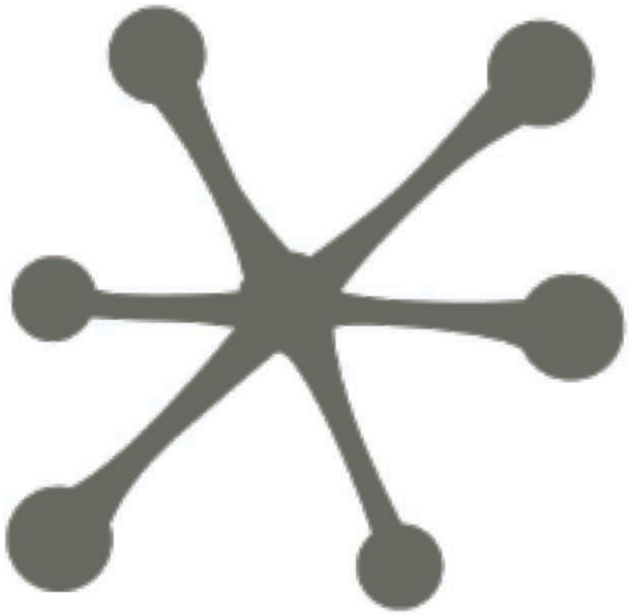
64bits Quad Core 1.2Ghz

17 GPIO (I2C, UART, ...)

Wifi

Compute unit

Tools



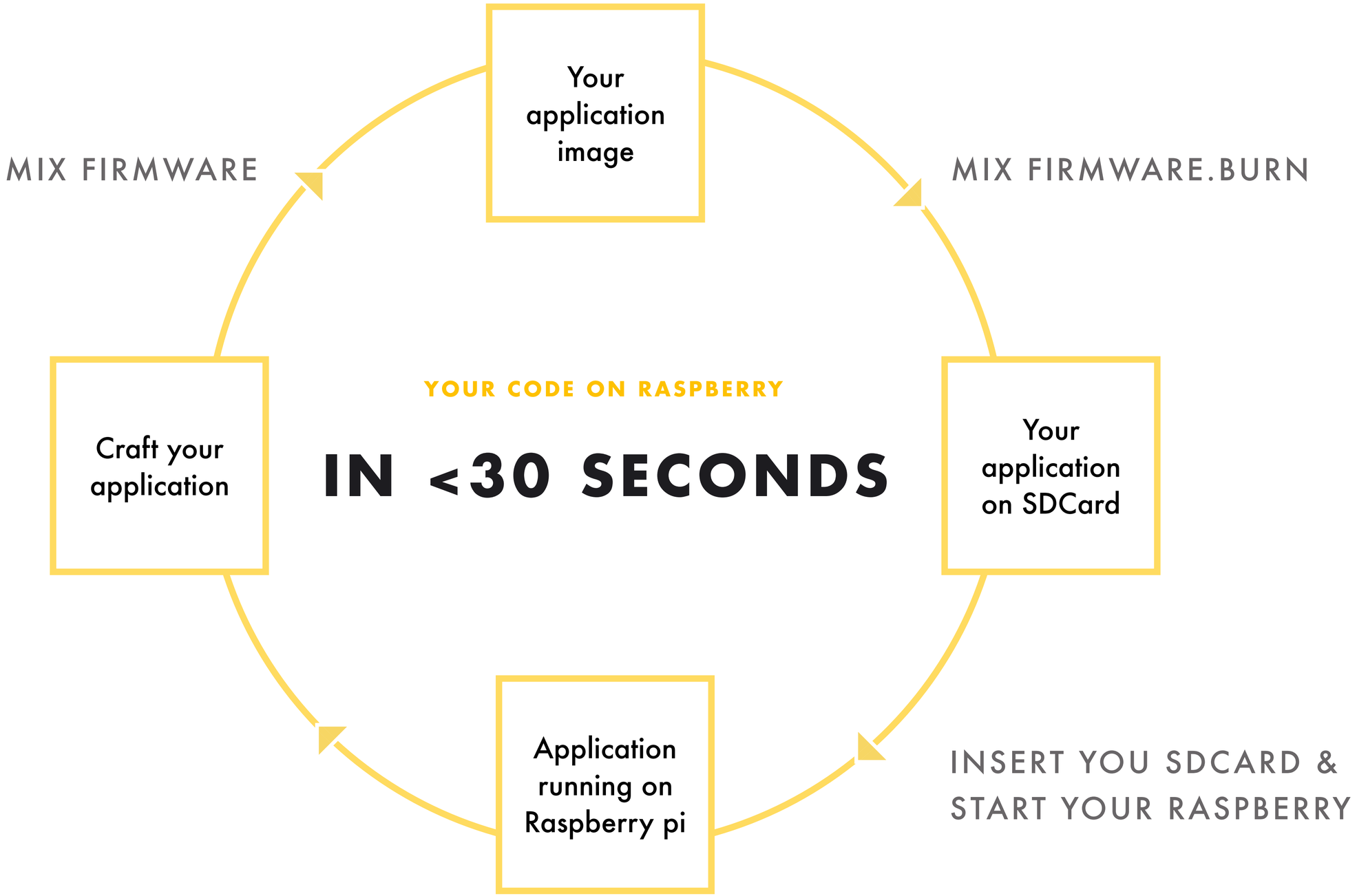
Nerves

Frameworks : Handle networks, I/O, ...

Tooling : Cross compiling tools for specific platform

Platforms : boot cross compiled linux directly to Erlang VM

DESIGN CHOICES



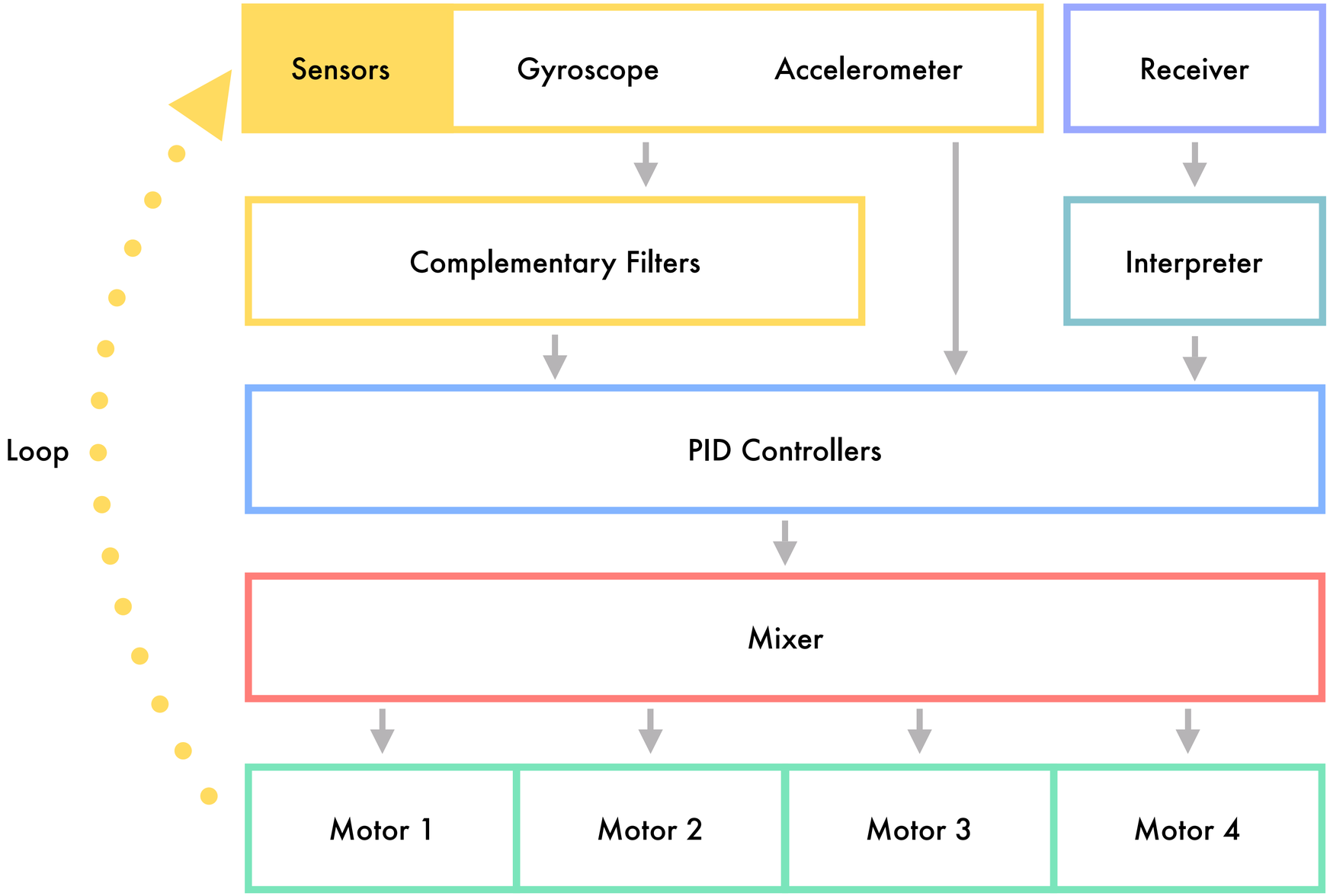
DESIGN CHOICES

BUILDING BLOCKS

EVERYTHING TOGETHER

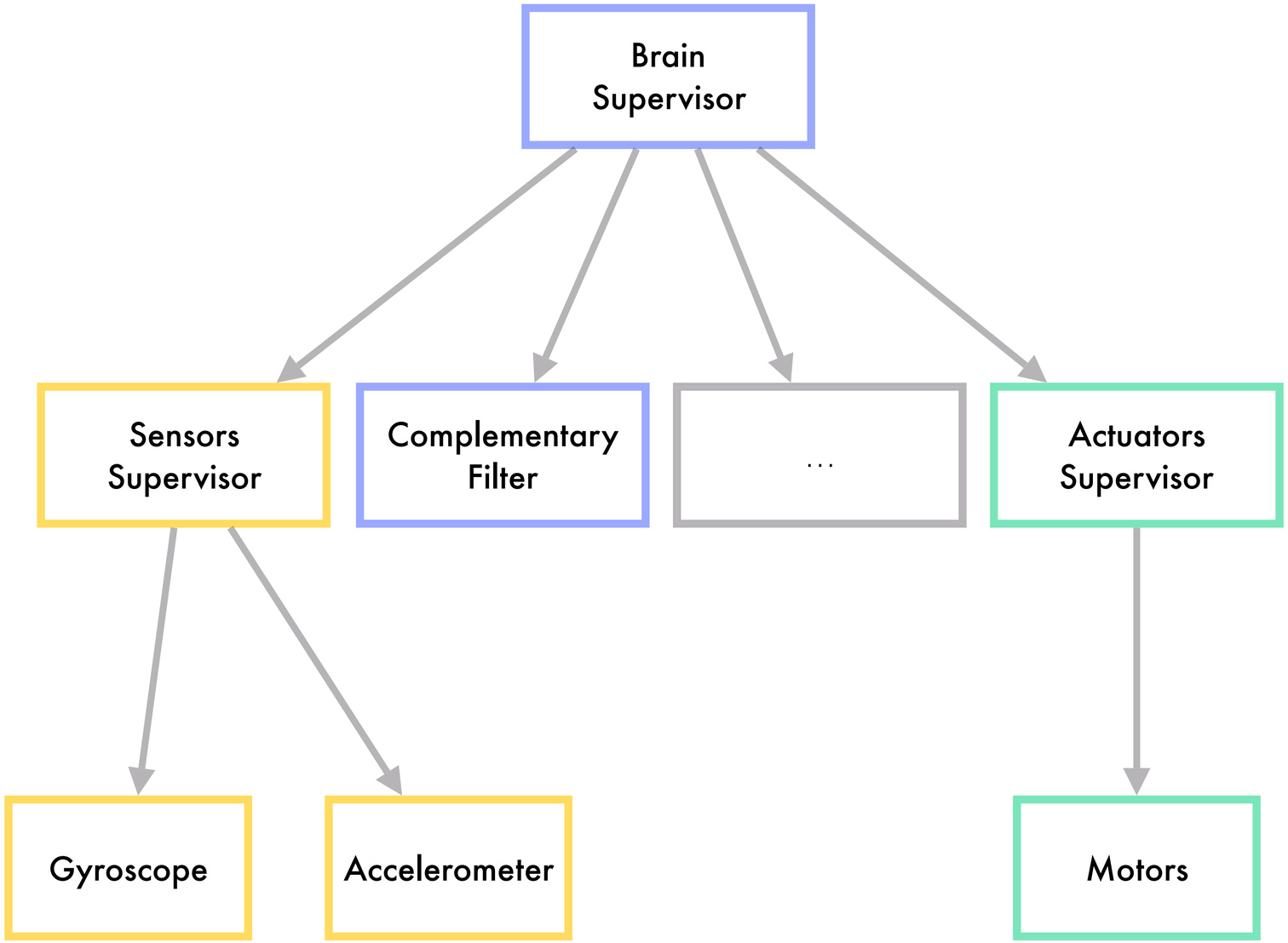
FURTHER WORK

BUILDING BLOCKS



IT'S JUST A TREE

- Each component is a GenServer
- Components grouped under supervisors



```
config :brain, :sensors, [
  Brain.Sensors.Magnetometer,
  Brain.Sensors.Accelerometer,
  Brain.Sensors.Gyroscope
]
config :brain, Brain.Sensors.Gyroscope,
  driver: Drivers.L3GD20H
```

config.exs

sensor_supervisor.ex

```
defmodule Brain.Sensors.Supervisor do
  use Supervisor

  def init([]) do
    children = Application.get_env(:brain, :sensors) |> Enum.map(fn sensor_module ->
      sensor_configuration = Application.get_env(:brain, sensor_module)
      driver_module        = sensor_configuration[:driver]
      worker(sensor_module, [driver_module])
    end)
    supervise(children, strategy: :one_for_one)
  end
end
```

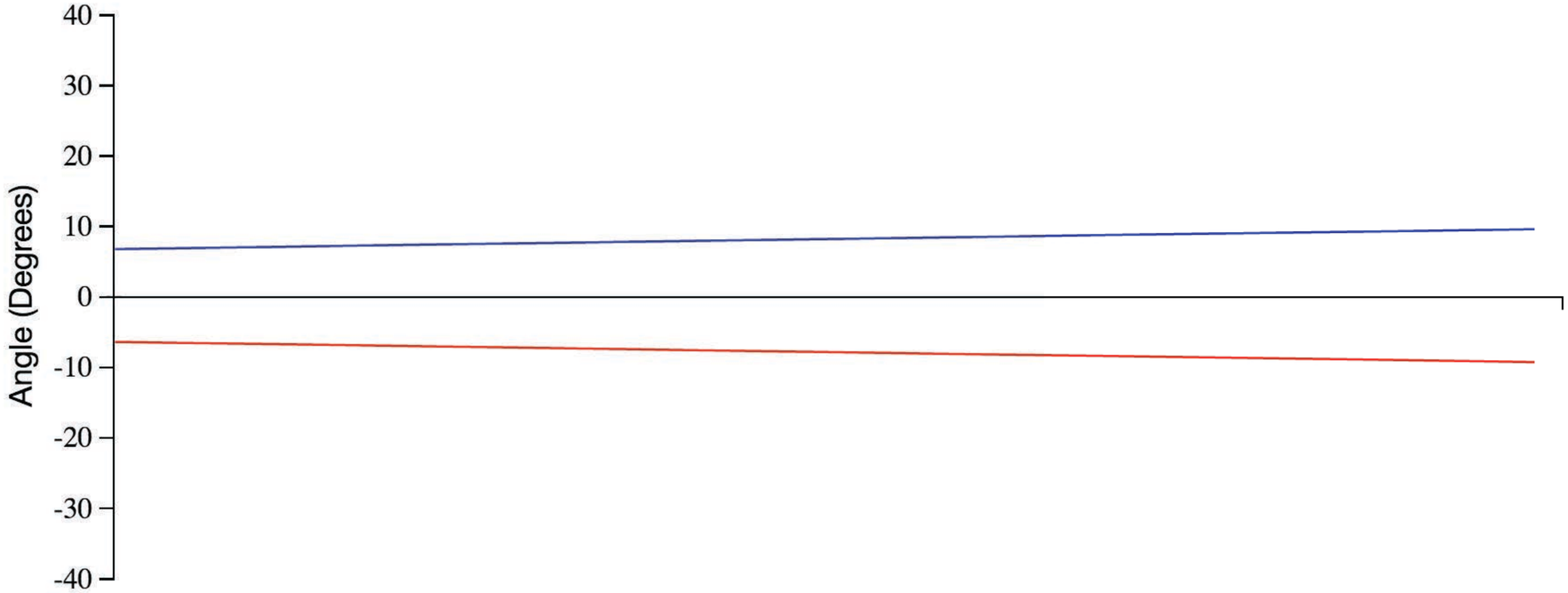
SENSORS

GET THE QUADCOPTER TILT

GYROSCOPE

Measures the angular speed (degrees/seconds)

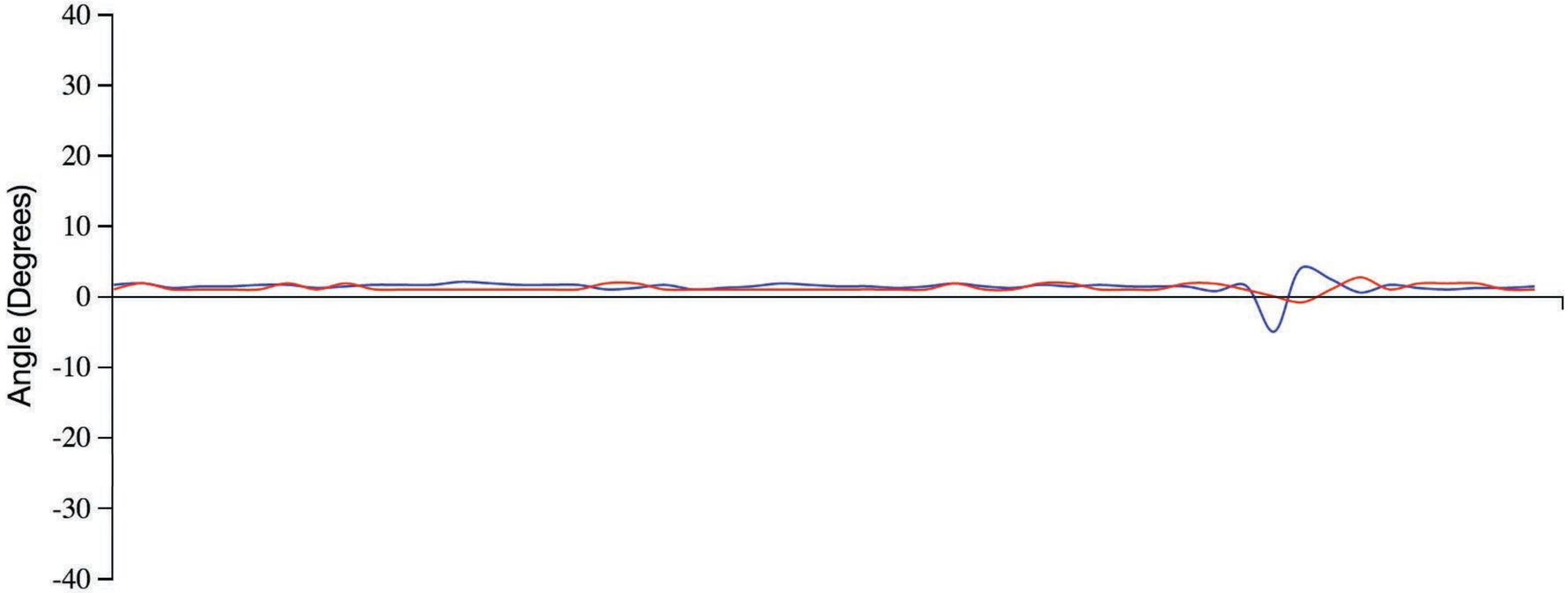
Its values drift with time



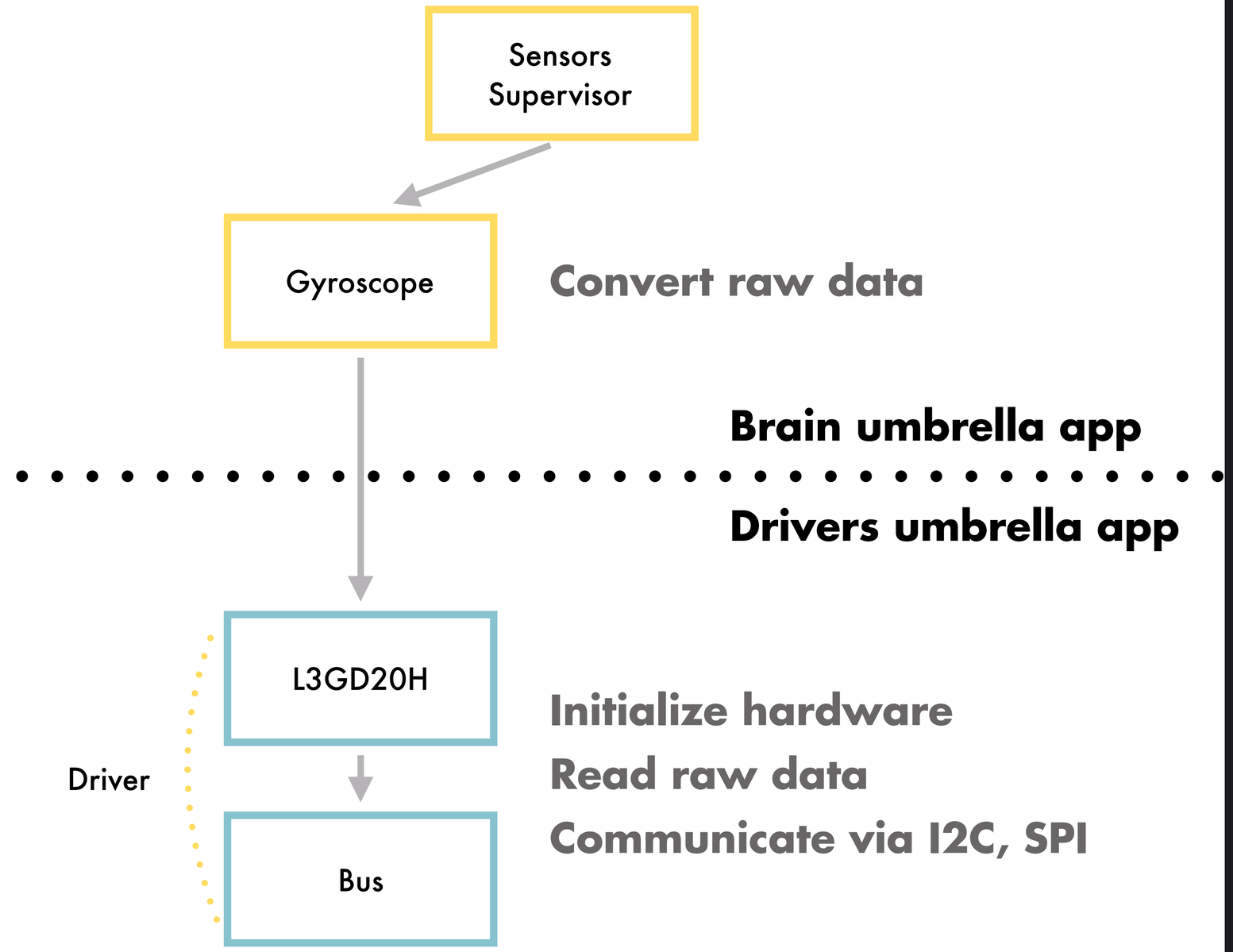
ACCELEROMETER

Measures the acceleration (g)

It is highly sensible to vibrations



BUILDING BLOCKS



IN PRACTICE

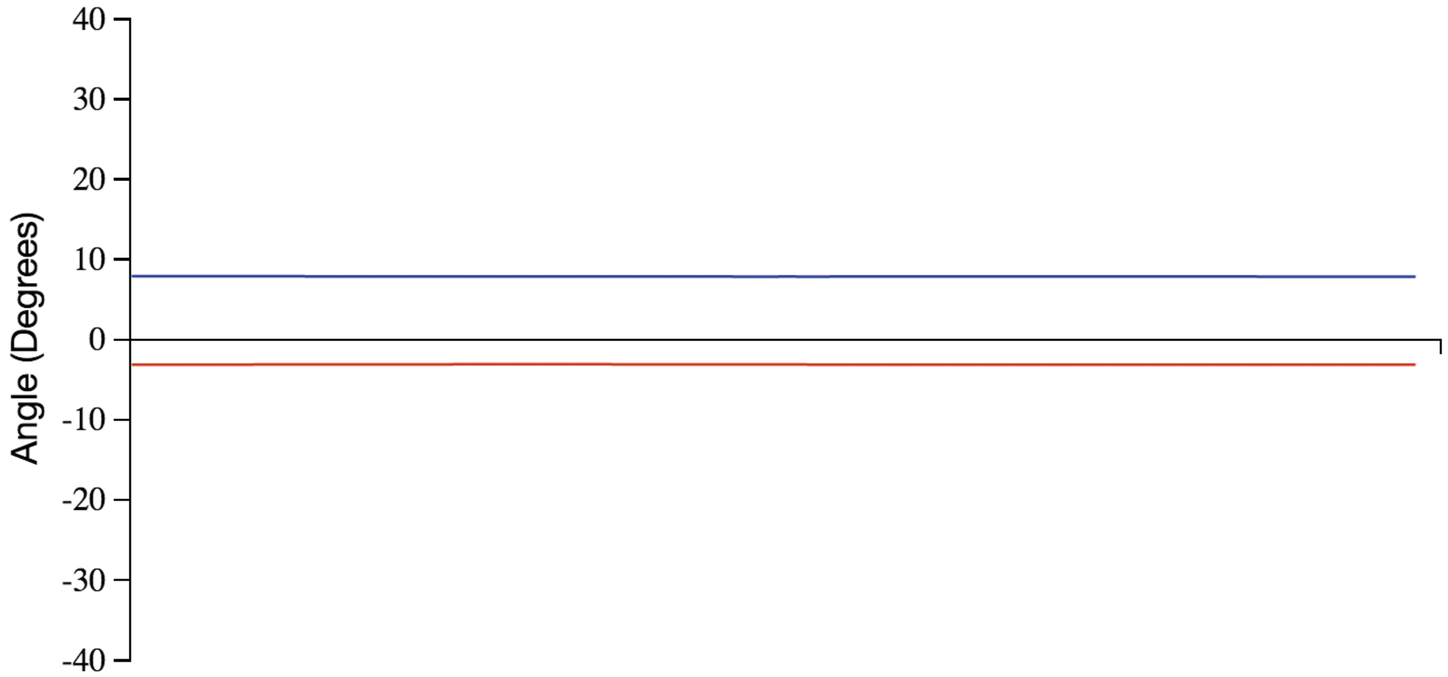
- Separate drivers and business logic
- Allow to switch driver for each sensor
- **elixir_ale** to handle communication

COMPLEMENTARY FILTER

Computes precise measure of the quadcopter tilt

Compensates accelerometer noise and gyroscope drift

angle = α * (gyroscope) + (1 - α) * accelerometer



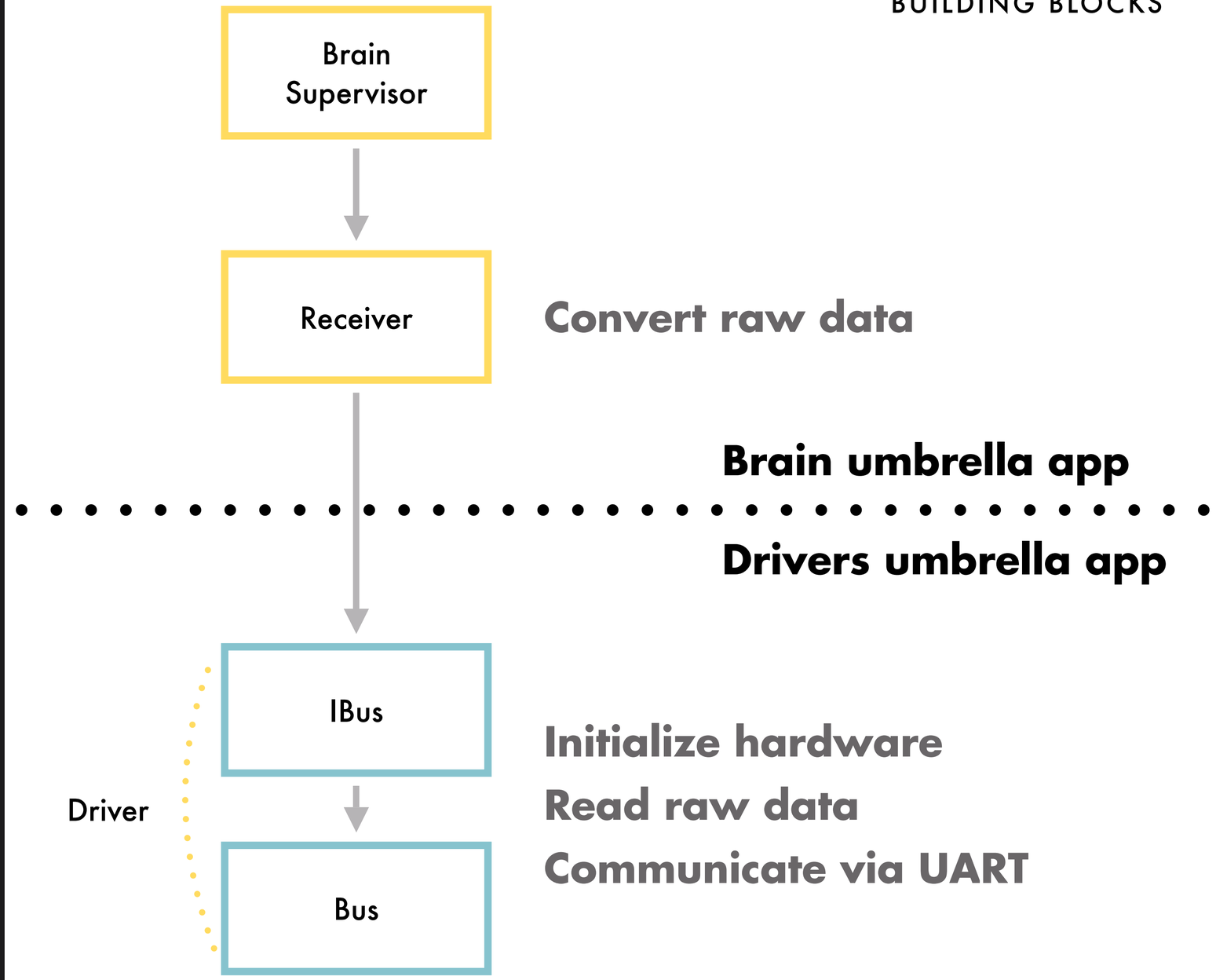
RECEIVER

RECEIVES COMMAND SENT BY USER TRANSMITTER

<<6, 0, 0, 1, 0, 0, 0, 255, 0, 0, 0, 2, 0, 0, 0, 255, 255>>

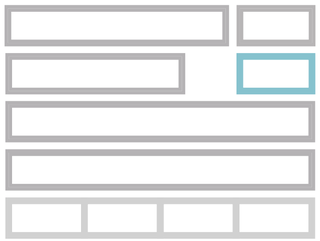
IN PRACTICE

- Same logic as sensors
- **nerves_uart** for serial communication

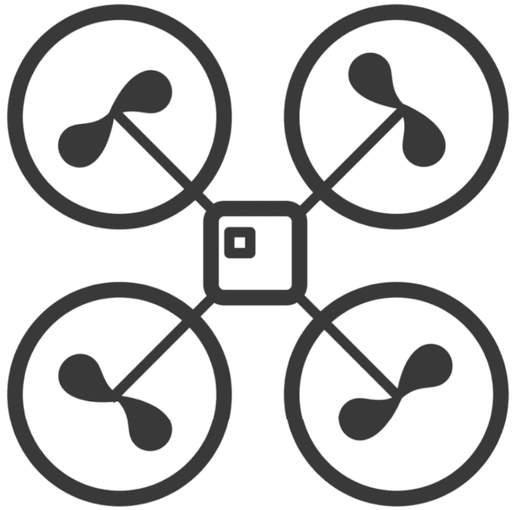


INTERPRETER

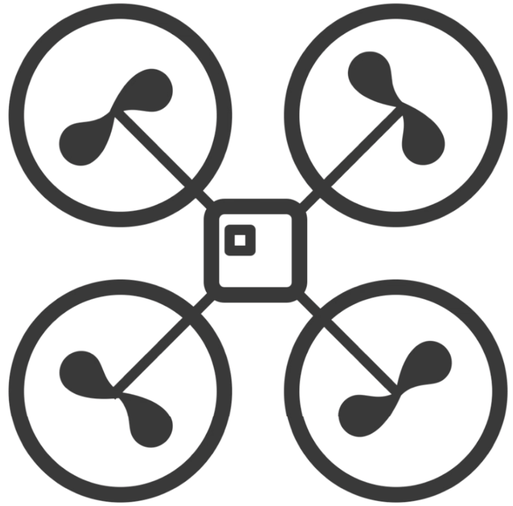
CONVERTS RECEIVER CHANNELS VALUES TO SET POINTS



RATE MODE

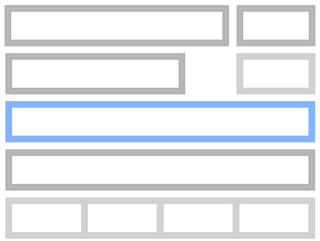


ANGLE MODE

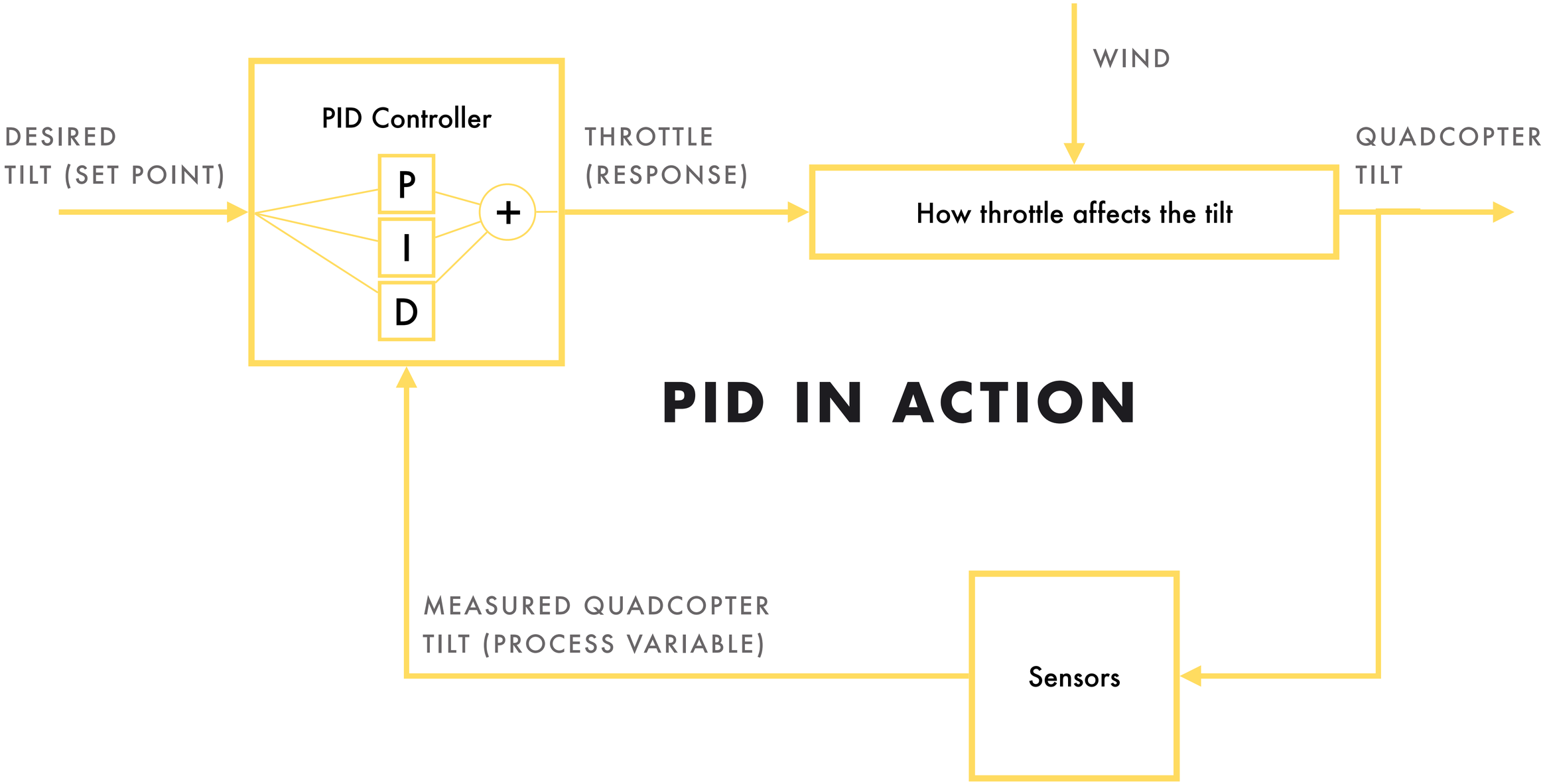


PID CONTROLLER

CONTROL LOOP FEEDBACK MECHANISM



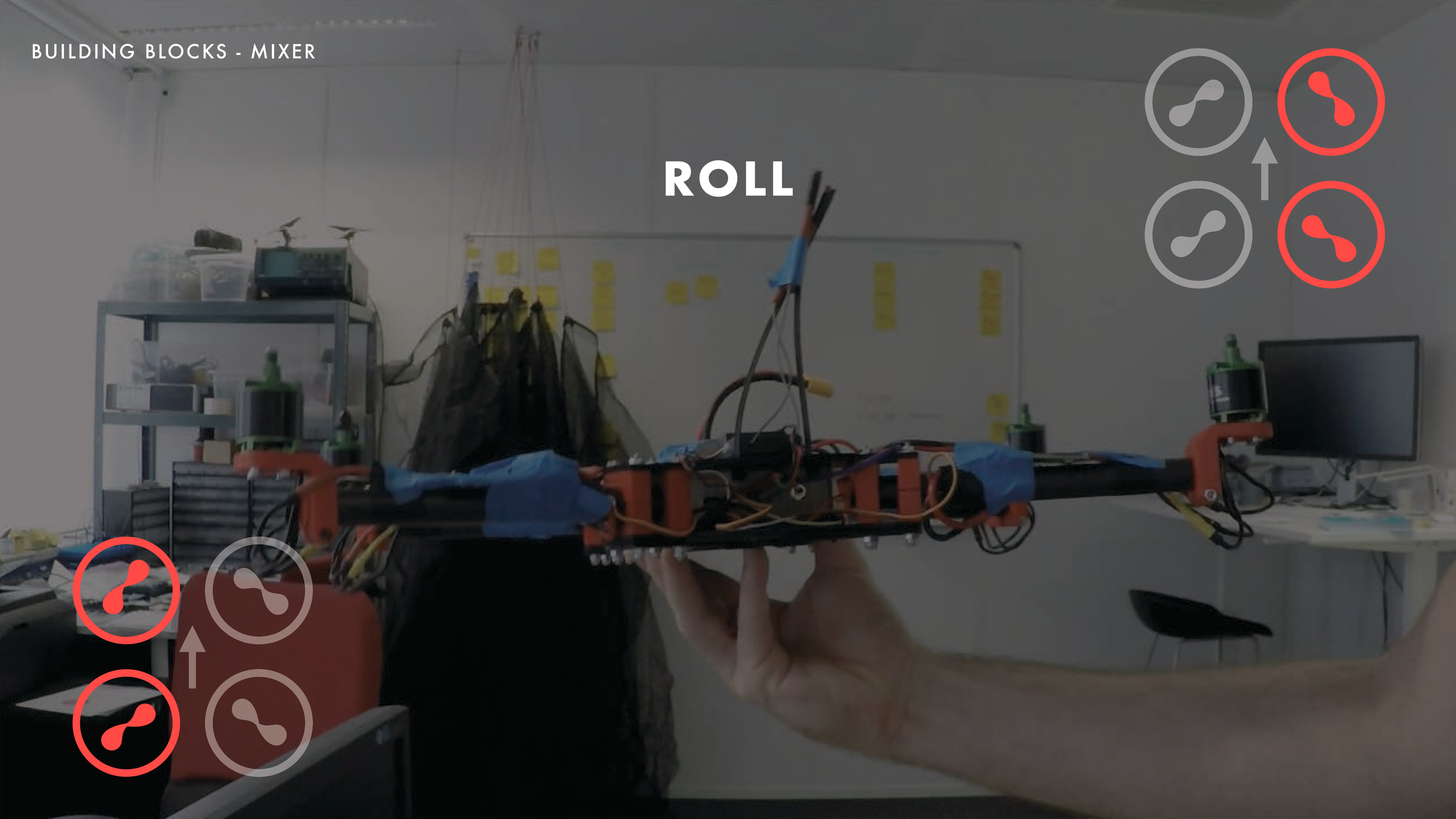
BUILDING BLOCKS - PID CONTROLLER



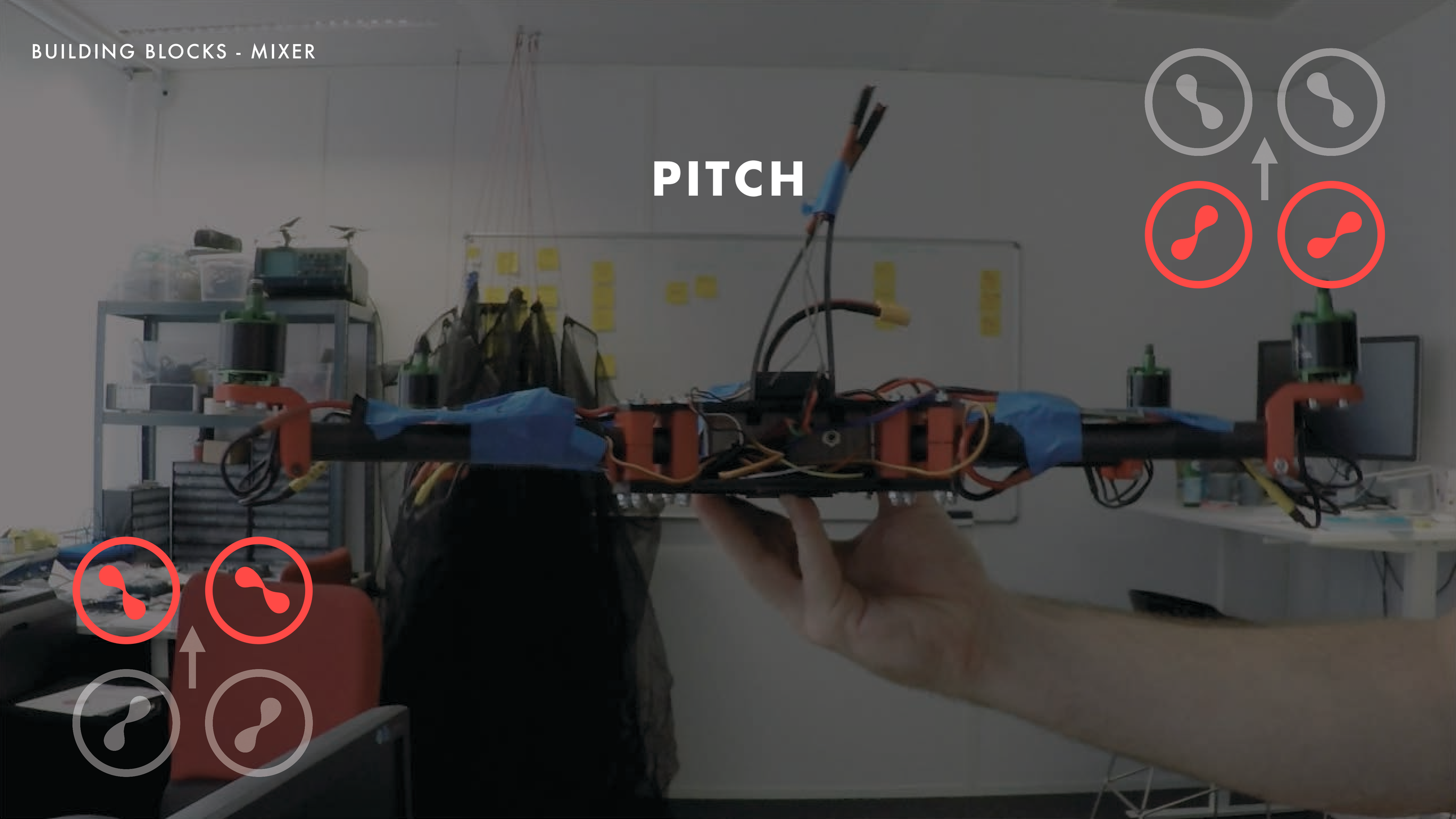
MIXER

COMPUTES THE SPEED TO APPLY TO EACH MOTOR

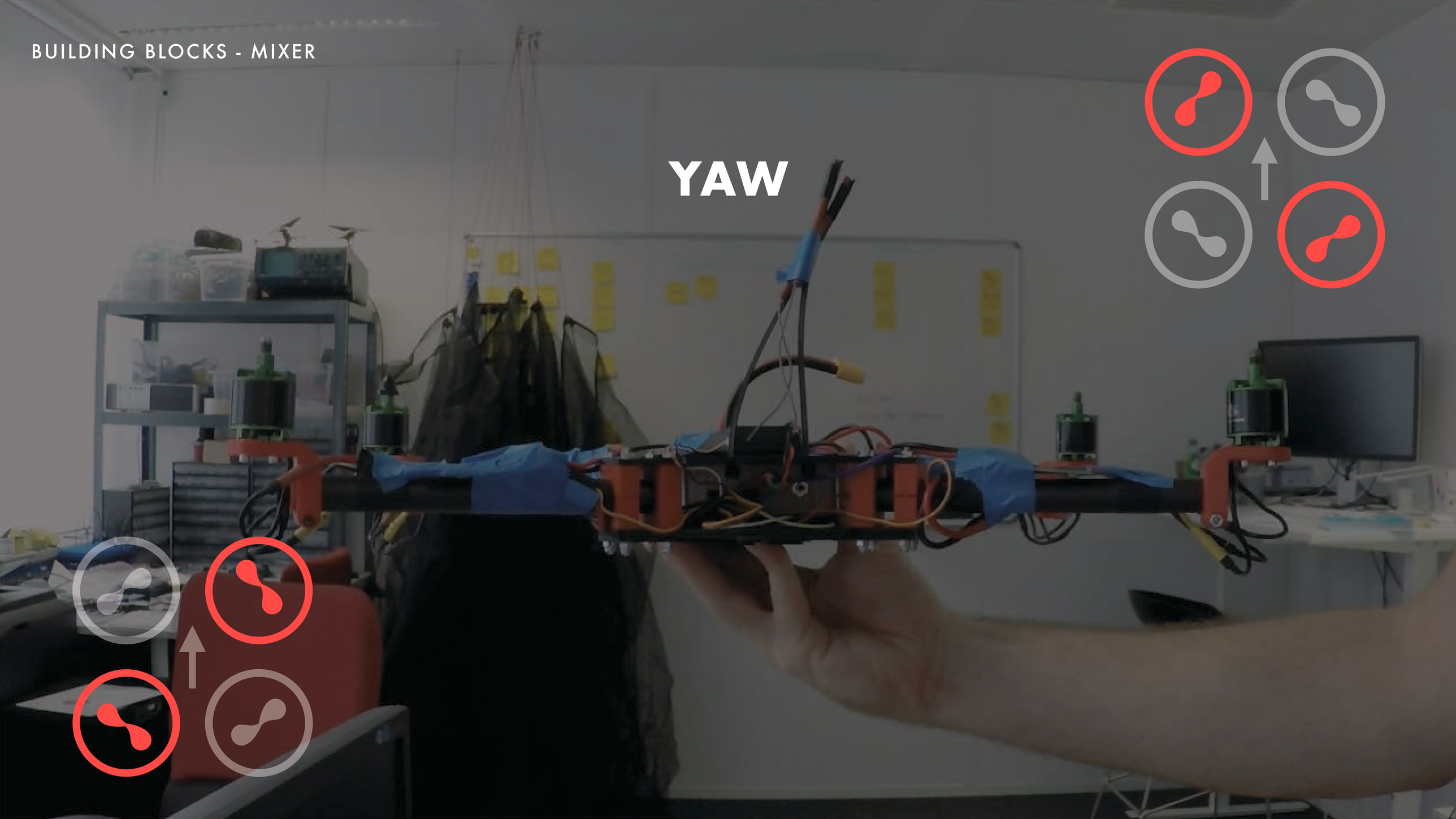
ROLL



PITCH



YAW



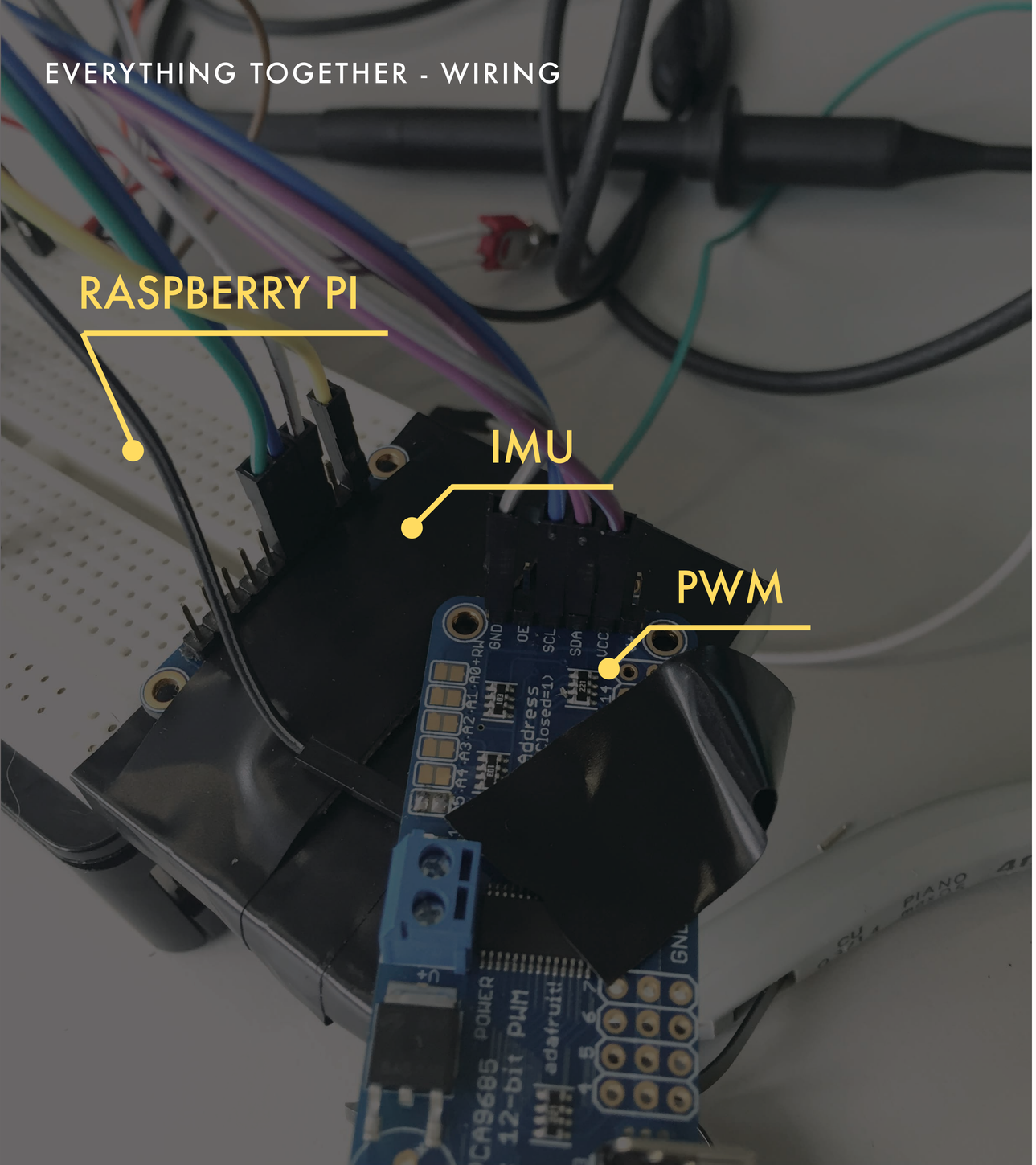
DESIGN CHOICES

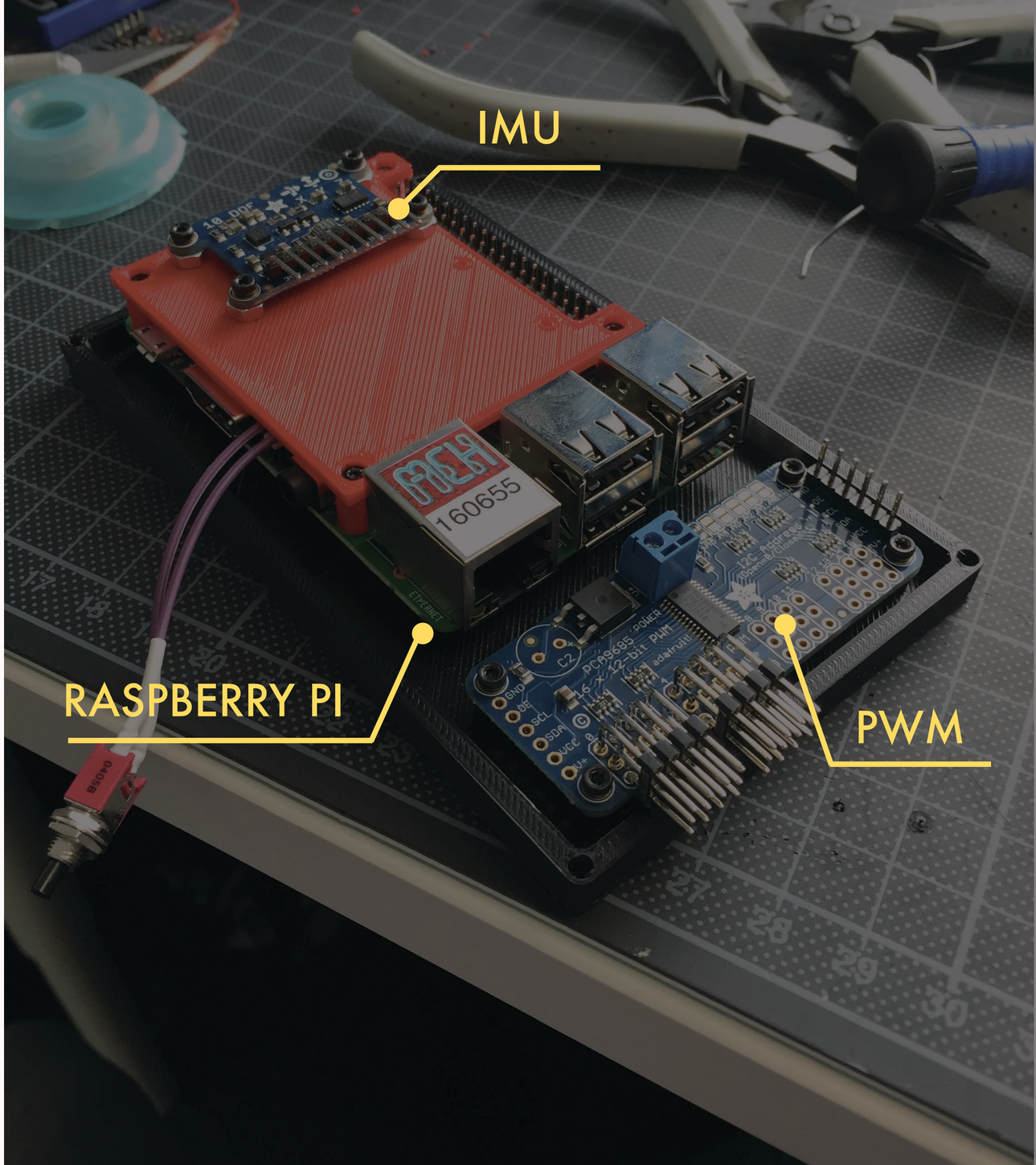
BUILDING BLOCKS

EVERYTHING TOGETHER

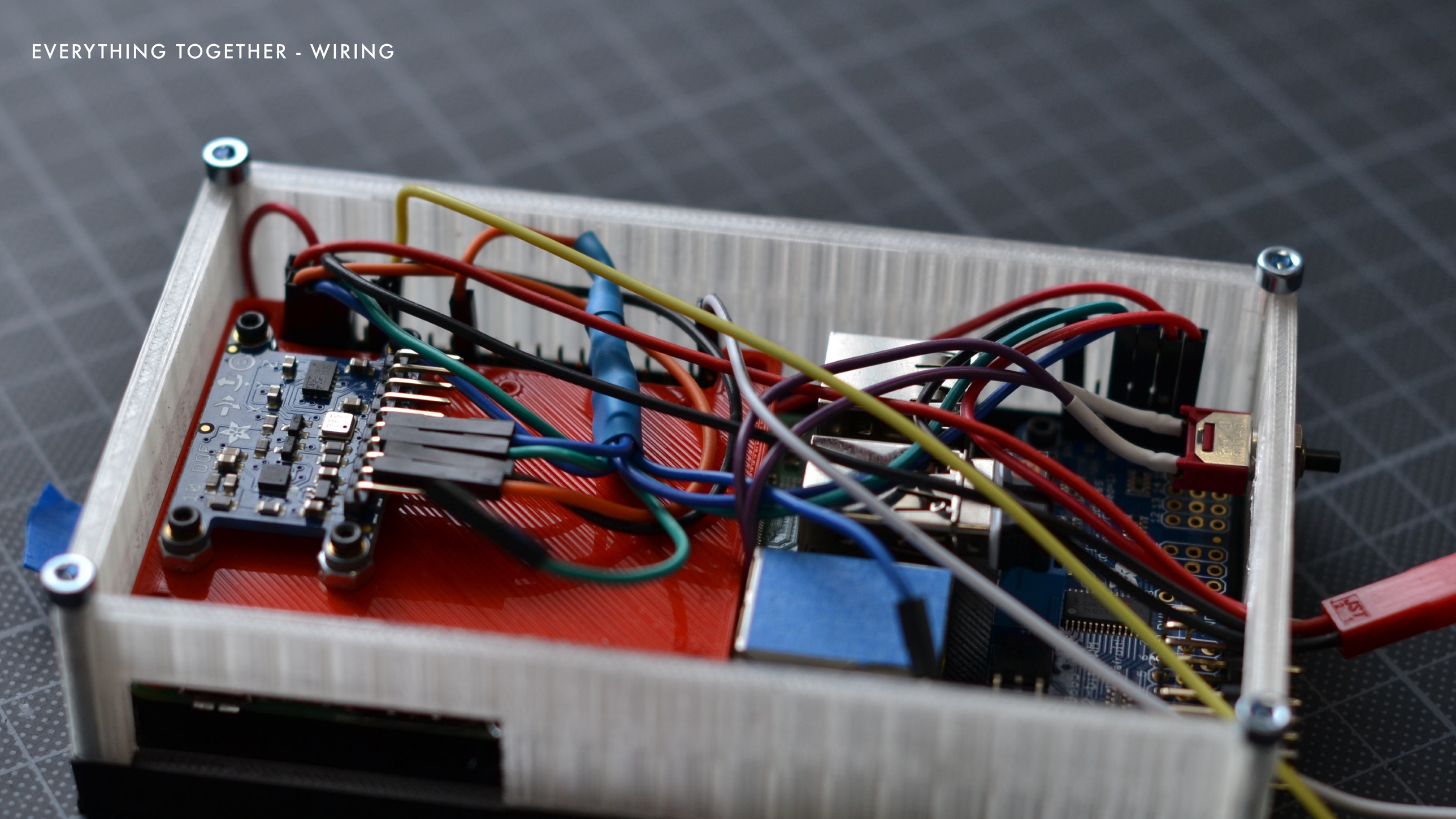
FURTHER WORK

EVERYTHING TOGETHER - WIRING

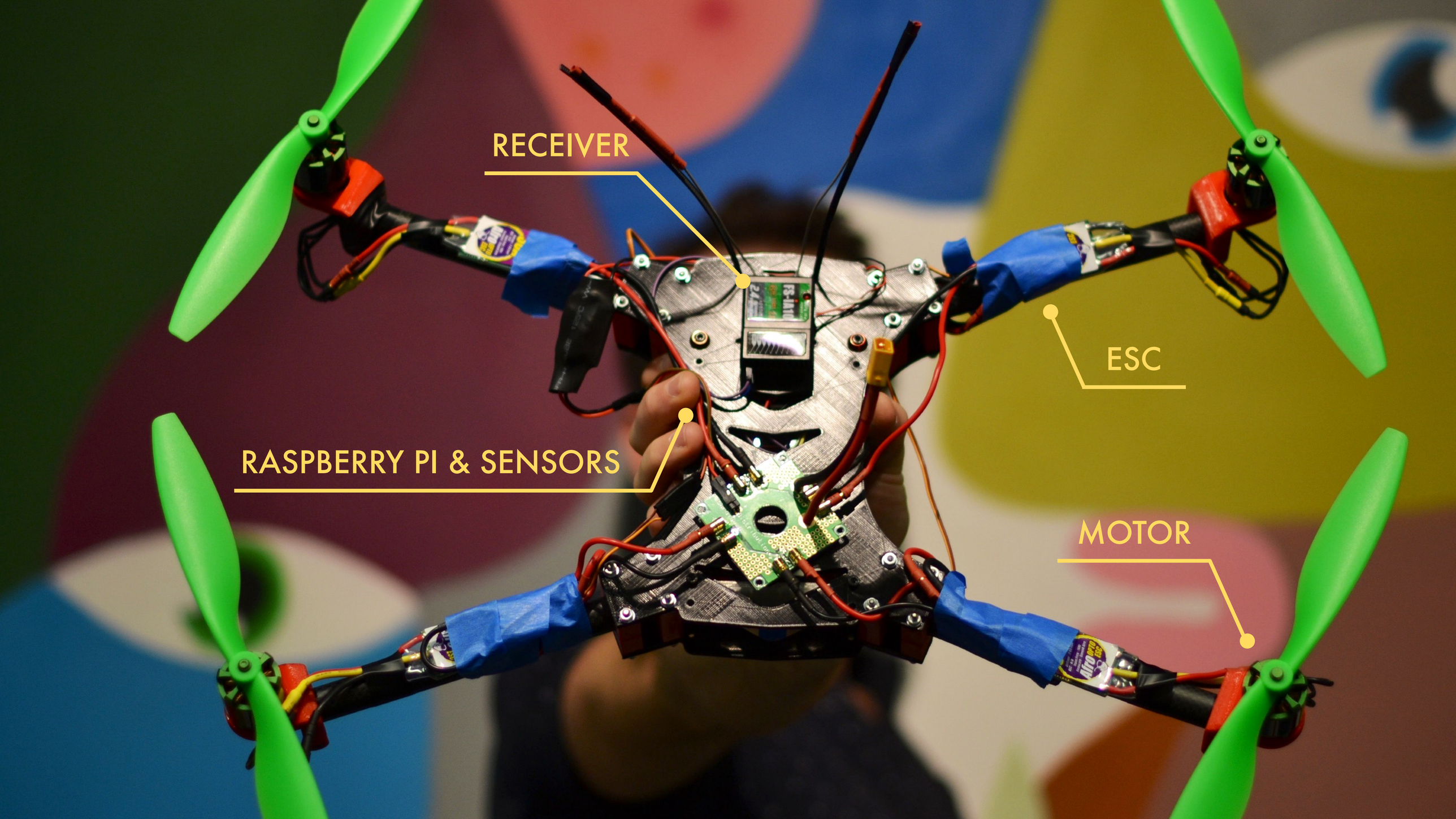


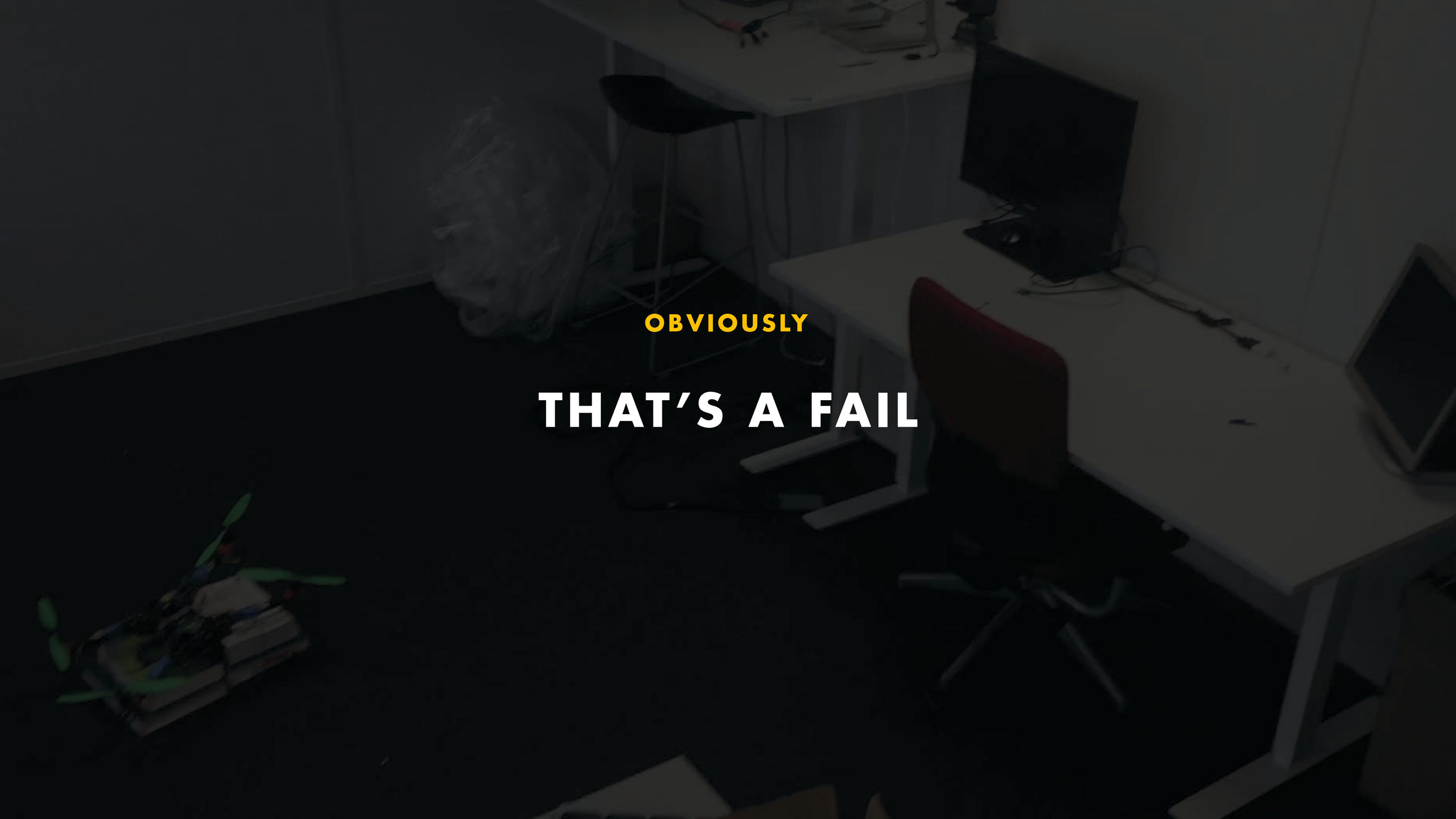


EVERYTHING TOGETHER - WIRING



EVERYTHING TOGETHER - WIRING


RECEIVER
ESC
RASPBERRY PI & SENSORS
MOTOR



OBVIOUSLY

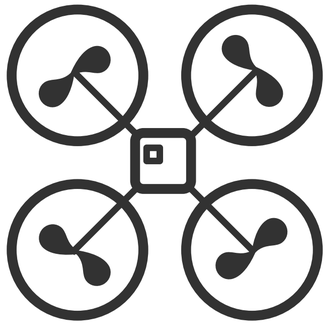
THAT'S A FAIL

DEBUGGING

REDUCE THE FEEDBACK LOOP

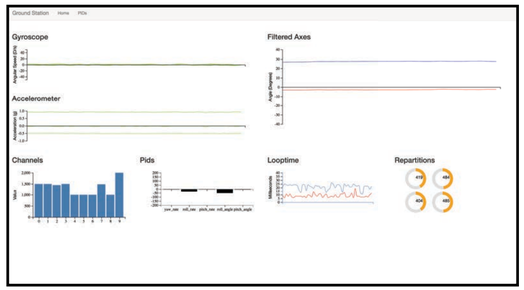
BLACKBOX

Collects data of each quadcopter components



Quadcopter

Phoenix Socket



Ground Station

Quadcopter Sensor

```
def handle_call(:read, _from, %{driver_pid: driver_pid} = state) do
  {:ok, data} = GenServer.call(driver_pid, :read)
  BlackBox.trace(__MODULE__, Process.info(self())[:registered_name], data)
  {:reply, {:ok, data}, state}
end
```

1

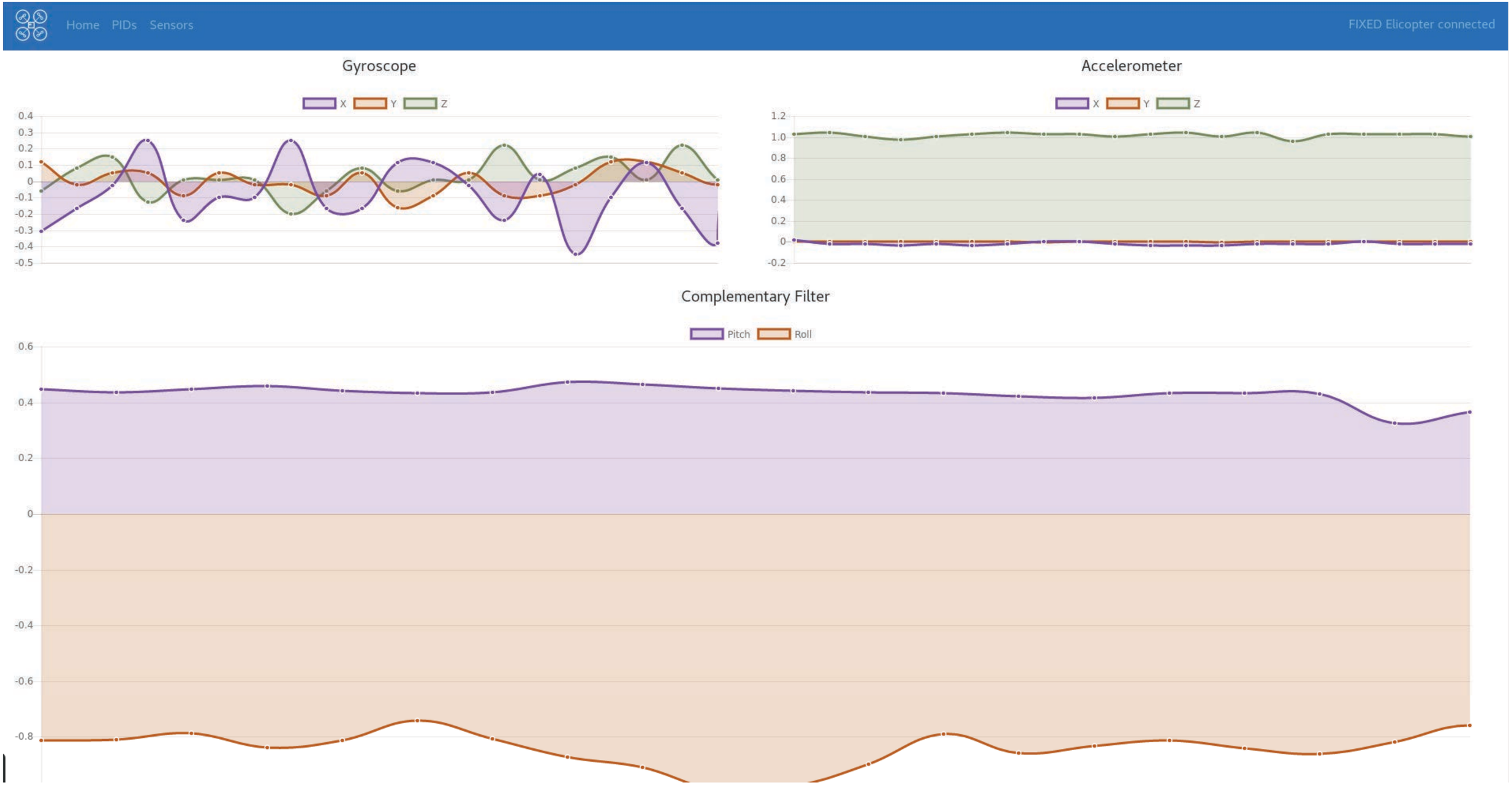
Blackbox

```
def handle_cast({:trace, key, data}, %{loops_buffer: loops_buffer} = state) do
  [last_loop | previous_loops] = loops_buffer
  last_loop                    = [{key, data} | last_loop]
  if length(previous_loops) >= @loops_buffer_limit do
    previous_loops = previous_loops |> Enum.drop(-1)
  end
  {:noreply, %{state | loops_buffer: [last_loop | previous_loops]}}
end

def trace(module, process_name, data) do
  event = case {module, process_name, data} do
    {Brain.Loop, _process_name, data} ->
      {:trace, :loop, data}
    {_, process_name, data} ->
      {:trace, process_name |> Module.split |> List.last |> Macro.underscore, data}
  end
  GenServer.cast(__MODULE__, event)
end
```

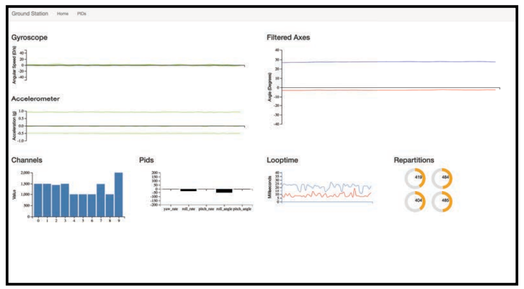
2

EVERYTHING TOGETHER - BLACKBOX



API

Tunes quadcopter in real time



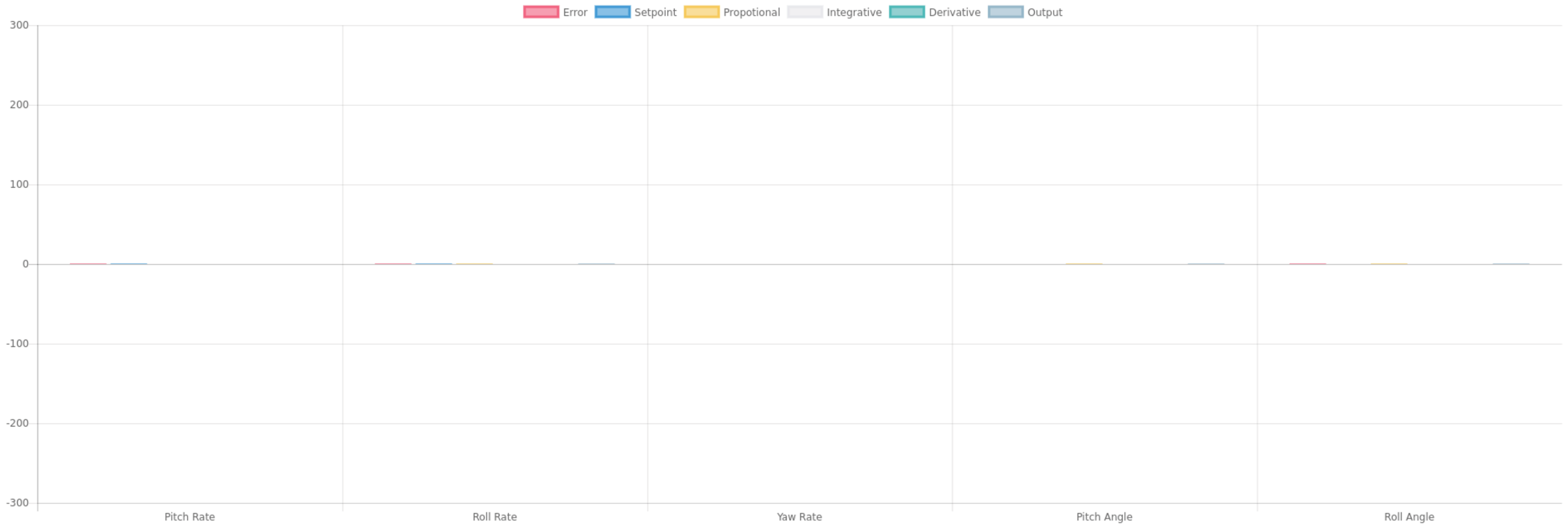
Ground Station

Phoenix REST API

Quadcopter

EVERYTHING TOGETHER - COMMANDER

Home PIDs Sensors FIXED Elicopter connected



Controllers

Name	kp			ki			kd			Minimum Output	Maximum Output
RollAnglePIDController	-	1.5	+	-	0	+	-	0	+	-400	400
PitchAnglePIDController	-	-1.5	+	-	0	+	-	0	+	-400	400
YawRatePIDController	-	1.7	+	-	0	+	-	0	+	-500	500
RollRatePIDController	-	0.7	+	-	0	+	-	0	+	-500	500
PitchRatePIDController	-	0.7	+	-	0	+	-	0	+	-500	500

SSDP DISCOVERY

- No need to change the IP address in the code
- **nerves_ssdp_server**
- **nerves_ssdp_client**

Quadcopter

1

```
def advertise_ssdp do
  name = Application.get_env(:brain, :name)
  {:ok, _name} = Nerves.SSDPServer.publish(name, "elicopter",[
    port: Application.get_env(:api, Api.Endpoint)[:http][:port]
  ])
end
```

Host

```
defp discover do
  Logger.info("Discovering Elicopters...")
  nodes      = Nerves.SSDPClient.discover
  elicopters = nodes |> Enum.filter(fn ({_name, info}) ->
    case info do
      %{host: _, st: "elicopter"} -> true
      _ -> false
    end
  end)
  {:ok, elicopters}
end
```

2

HTTP FIRMWARE UPDATE

- Use SSDP discovery
- No need to plug or remove the SD Card
- Auto reboot
- **nerves_firmware_http**


```
defmodule Mix.Tasks.Firmware.Upgrade do
  use Mix.Task
  require Logger
  def run(_) do
    {:ok, _pid}        = HTTPoison.start
    {:ok, _pid}        = Nerves.SSDPClient.start(nil, nil)
    {:ok, elicopters} = discover()
    {elicopter_name, elicopter_info} = elicopters |> List.first
     Logger.info("Found #{elicopter_name} at #{elicopter_info[:host]}.")
    build()
    upload(elicopter_info[:host])
    Logger.info("Firmware updated!")
  end

  defp discover do
    …

  defp build do
    …
  end

  defp upload(host) do
    Logger.info("Upload firmware...")
    firmware_update_url = "http://#{host}:8988/firmware"
    firmware_path       = Path.absname("_images/rpi3/brain.fw")
    HTTPoison.request!(
      :post, firmware_update_url, {:file, firmware_path},
      [{"Content-Type", "application/x-firmware"}, {"X-Reboot", "true"}],
      [timeout: 30_000, recv_timeout: 30_000]
    )
  end
end
```

1

2

3

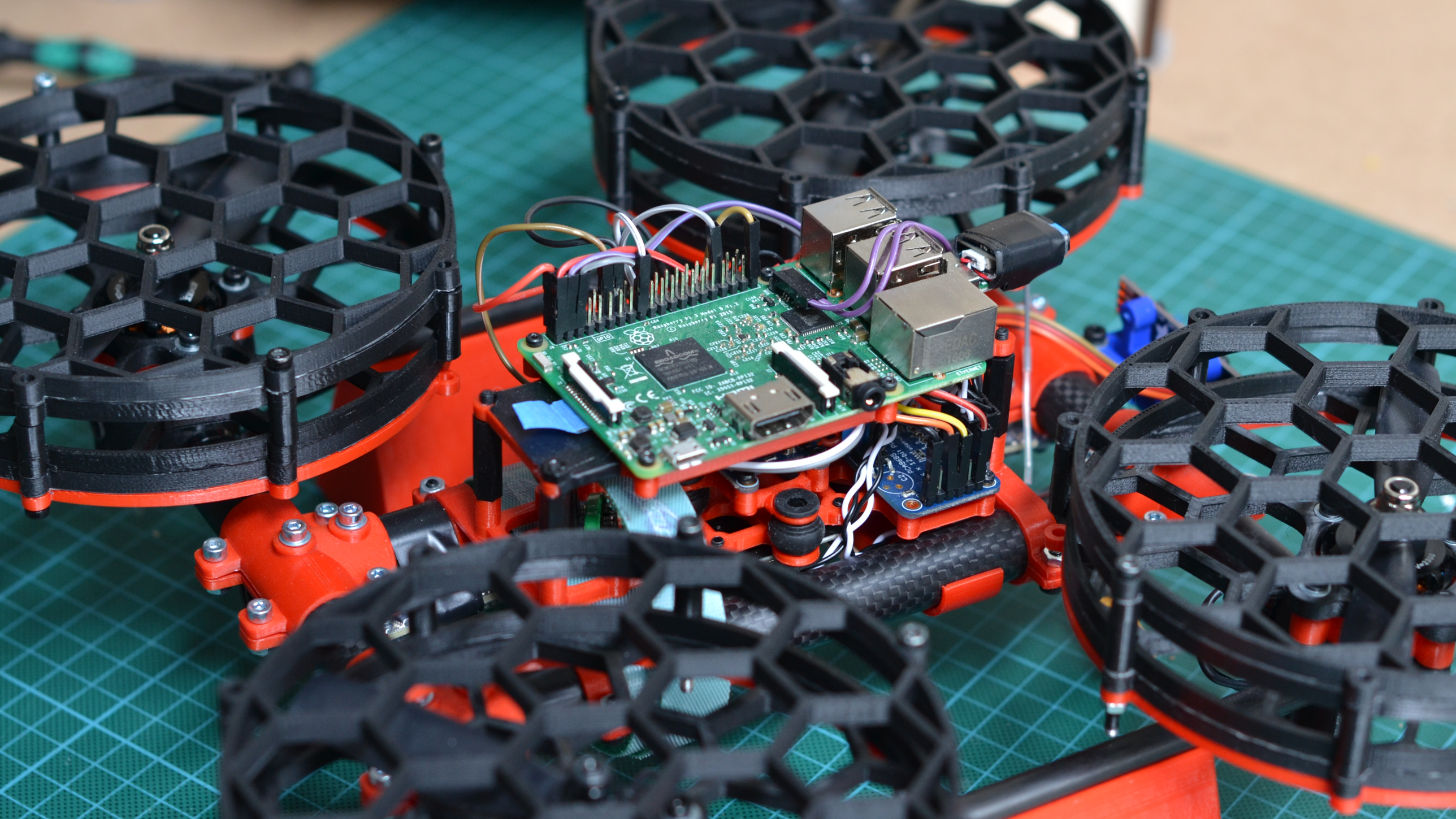
PREPARE A SAFE DEBUGGING ENVIRONMENT

THE RESULT

AFTER SOME TUNING...


VIC
TO
RY!


DEPRECATED



DESIGN CHOICES

BUILDING BLOCKS

EVERYTHING TOGETHER

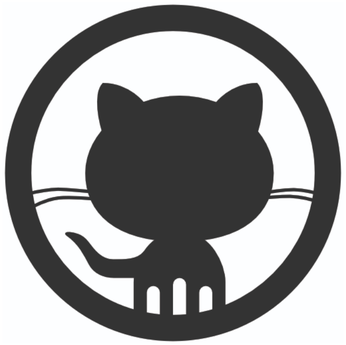
FURTHER WORK

IMPROVE

INNOVATE

OPEN SOURCE PROJECT

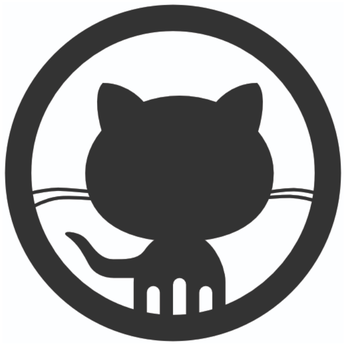
DRAWINGS AND CODE ON GITHUB



GITHUB.COM/ELICOPTER

ElixirConf™ EU

THANKS



GITHUB.COM/ELICOPTER

ElixirConf™ EU