

# **Hardware II Seminar**

Sensors and Data Analysis

# **Getting to know each other**

Óscar







GRUPO  
SANTANDER

At full power (1.31) there will be 2.74mA as Heating voltage that give us 37mA of Heating current and 10mW of Heating power.

Calculating PWM for typical Heating power:  
(Load resistor + Heating resistor) \* Heating current = desired voltage  
(15 + 74) \* 32 = 2.848V - 86.3% duty cycle since it is active LOW - 13.7% duty cycle

- **PWM resolution**

In theory we can use 10 or 12 bit resolution but maybe it has some impact on the possible frequency range, still to be checked

The Zero has the following hardware capabilities:

- 10 pins which default to 8-bit PWM (on the ATmega328P board). These can be mapped to 10-bit resolutions.
- 1 pin with 10-bit DAC (Digital-to-Analog Converter)

By setting the pins mode to 10, you can use analogWrite10() with values between 0 and 1023 to acquire the full DAC resolution.

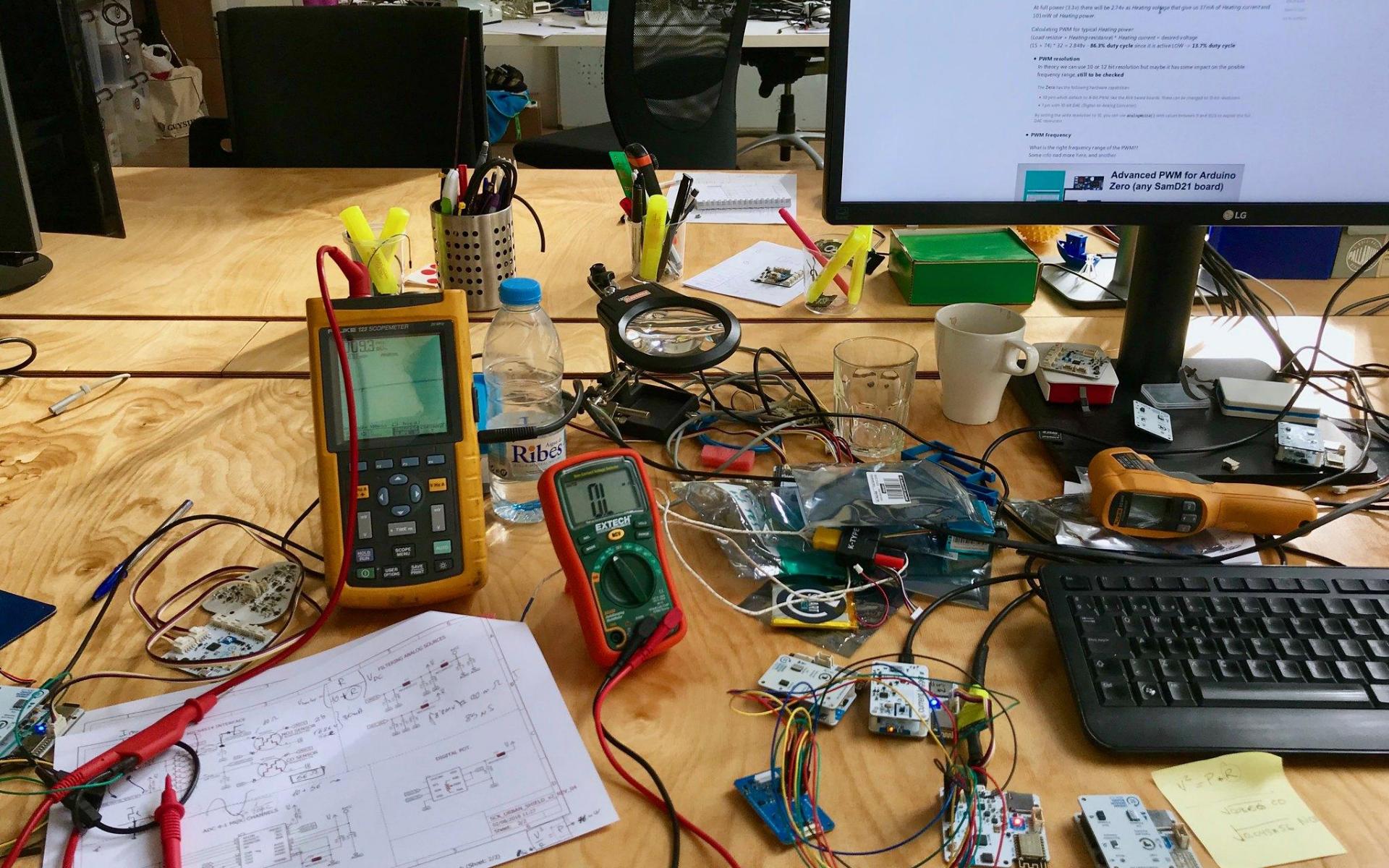
- **PWM frequency**

What is the right frequency range of the PWM?

Some info can be found here, and another

**Advanced PWM for Arduino Zero (any SamD21 board)**

LG





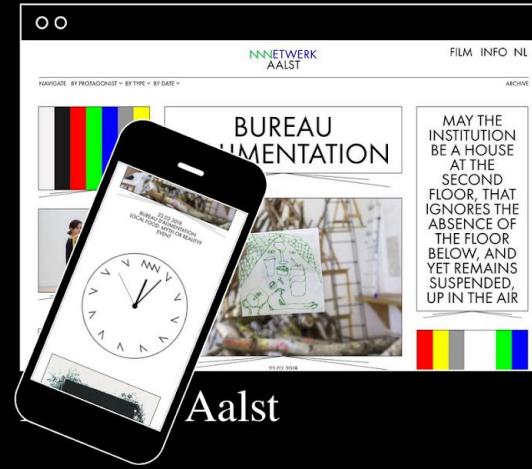
# **Getting to know each other**

Antoine

Variable est un studio de design et développement web spécialisé dans la création de sites et d'outils. Parcourez nos derniers projets et contactez-nous pour en savoir plus.



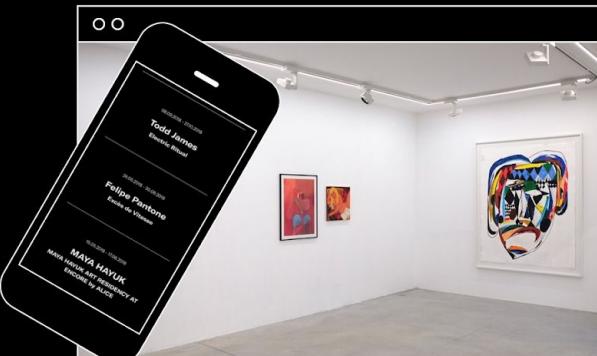
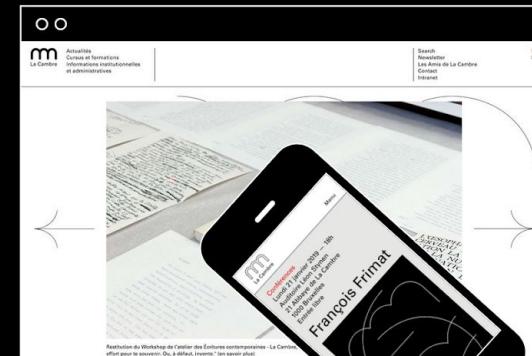
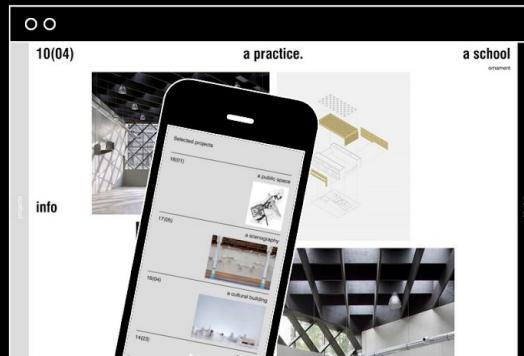
Éva Le Roi

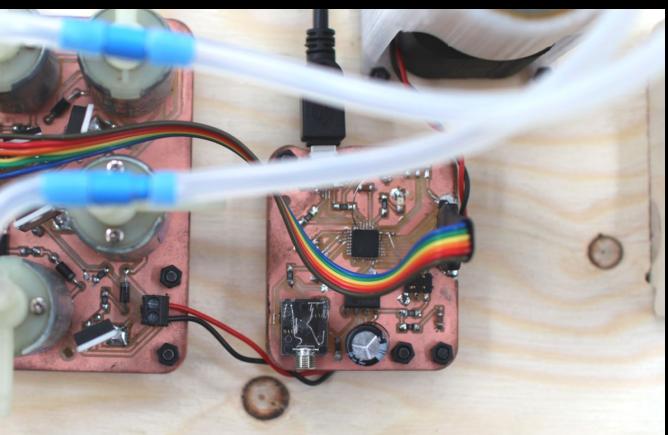
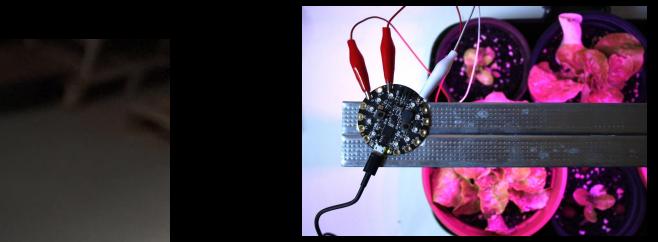
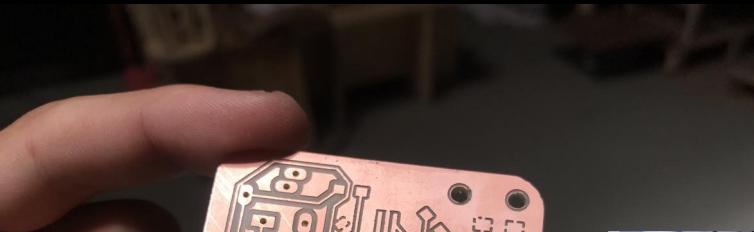


Aalst



Word Radio





<https://antoine.studio>



# **Getting to know each other**

You!

# **What we are going to do**

## Seminar Info

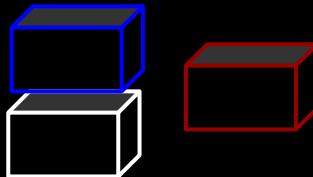
# What we are going to do - Course overview

- *Day 1: Introduction*
  - *Getting to know each other*
  - *Seminar info:*
    - *Overview and objectives*
    - *Exercise explanation*
  - *Inspiration*
  - *Sensor basics*
    - *Digitalising information*
    - *Sensor types*
    - *Sensor deployment scenarios*
- *Day 2: Making your own data-logger*
  - *Making an actual sensor setup*
    - *Use case 1: Arduino with sensor*
    - *Use case 2: RPI with digital sensor*
    - *Use case 3: Camera with various options*
    - *Use case 4: Multiple sensors network*
- *Day 3: Introduction to Data Analysis*
  - *1D DSP*
  - *2D DSP*
- *Day 4 Computer vision (Ángel Muñoz)*

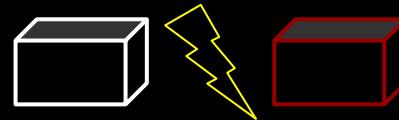
# What we are going to do - Course overview



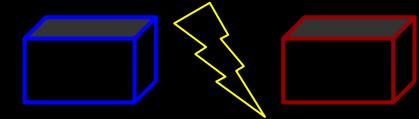
*Day 1: Sensors*



*Day 2: Sensors,  
cameras and  
Computers*



*Day 3: Same but useful*



*Day 4: Same but 2D  
sensors (cameras)*

## What we are going to do - Course objectives

- *Understanding different types of sensors and measurement principles*
- *Learn how to extract meaningful insights from different sensors using data analysis techniques*
- *Understand data logging processes in real time and with buffers*
- *Understand the trade-off between cost and complexity in the sensor selection*
- *More expensive is not always better*

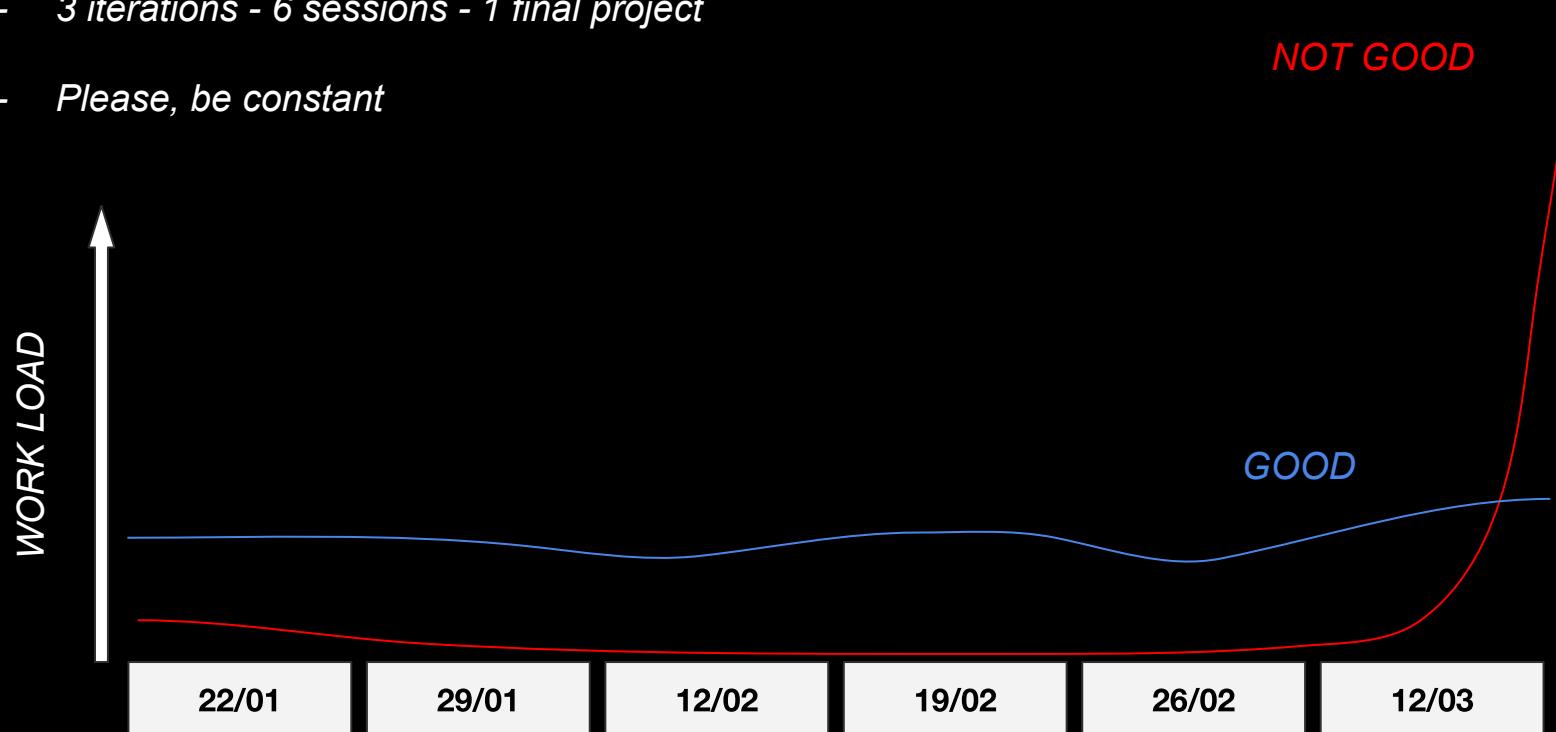
## What we are going to do - Exercise

- Groups of 2 to 3 people each. You decide.
- The exercise consists in creating a hardware setup and data acquisition system using at least one sensor type of the sensors we'll see during this week

22/01	Project idea presentation - potential conceptualization, plan and BOM
29/01	First <i>minimum viable product</i> prototype
12/02	Second iteration
19/02	Mid-term review
26/02	Third iteration
12/03	Final presentation - Back to life

## What we are going to do - How to go for it

- *3 iterations - 6 sessions - 1 final project*
- *Please, be constant*



# **Before we get started**

## Important resources

## Before we get started - Important information

- **Code repository:** [https://github.com/oscgonfer/sensors\\_dsp\\_lectures](https://github.com/oscgonfer/sensors_dsp_lectures)

```
cd /path/to/nice/mrac/documents/folder
```

```
git clone https://github.com/oscgonfer/sensors_dsp_lectures.git
```

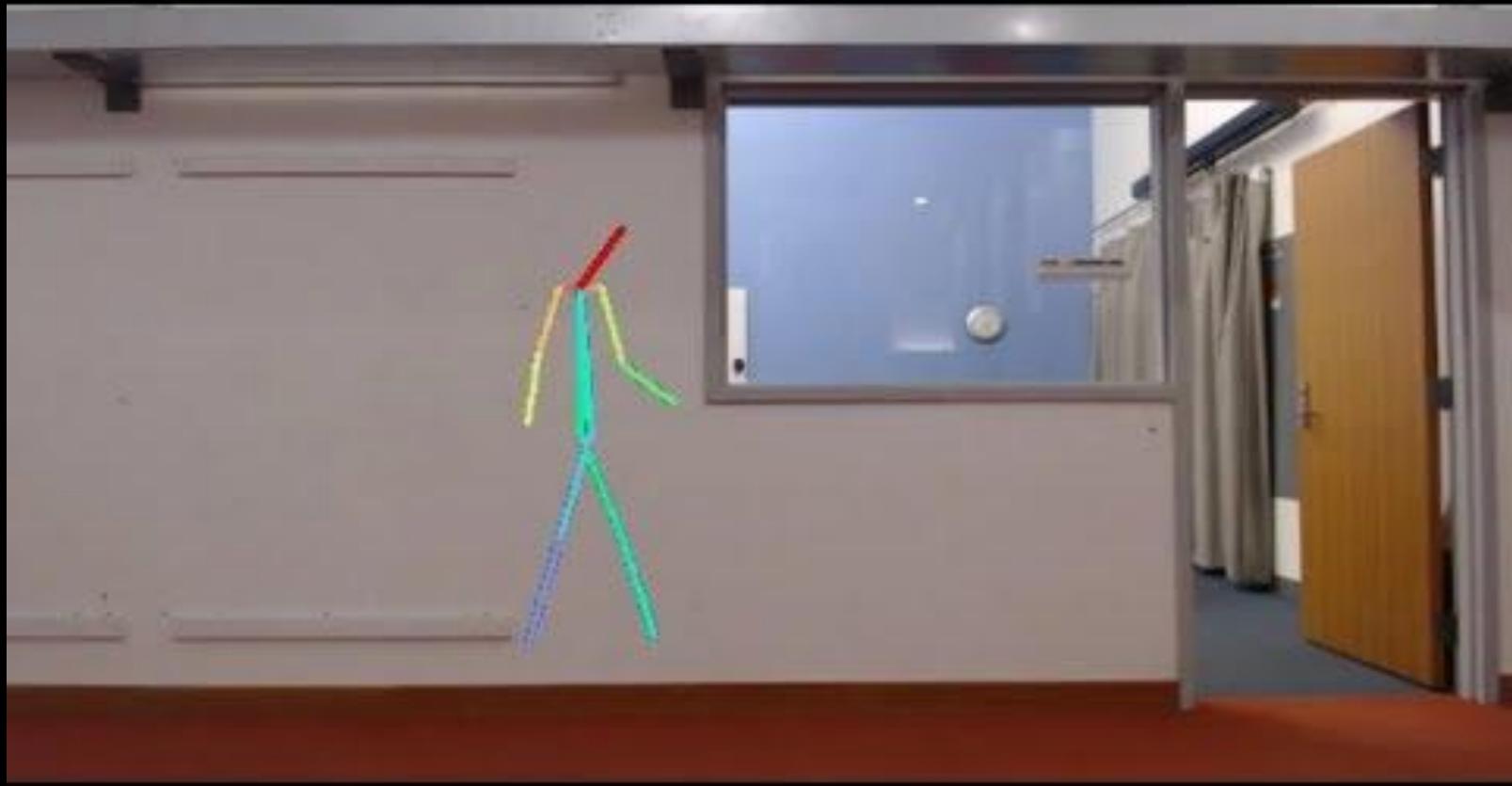
- **MRAC Group on Github:** <https://github.com/MRAC-IAAC/>

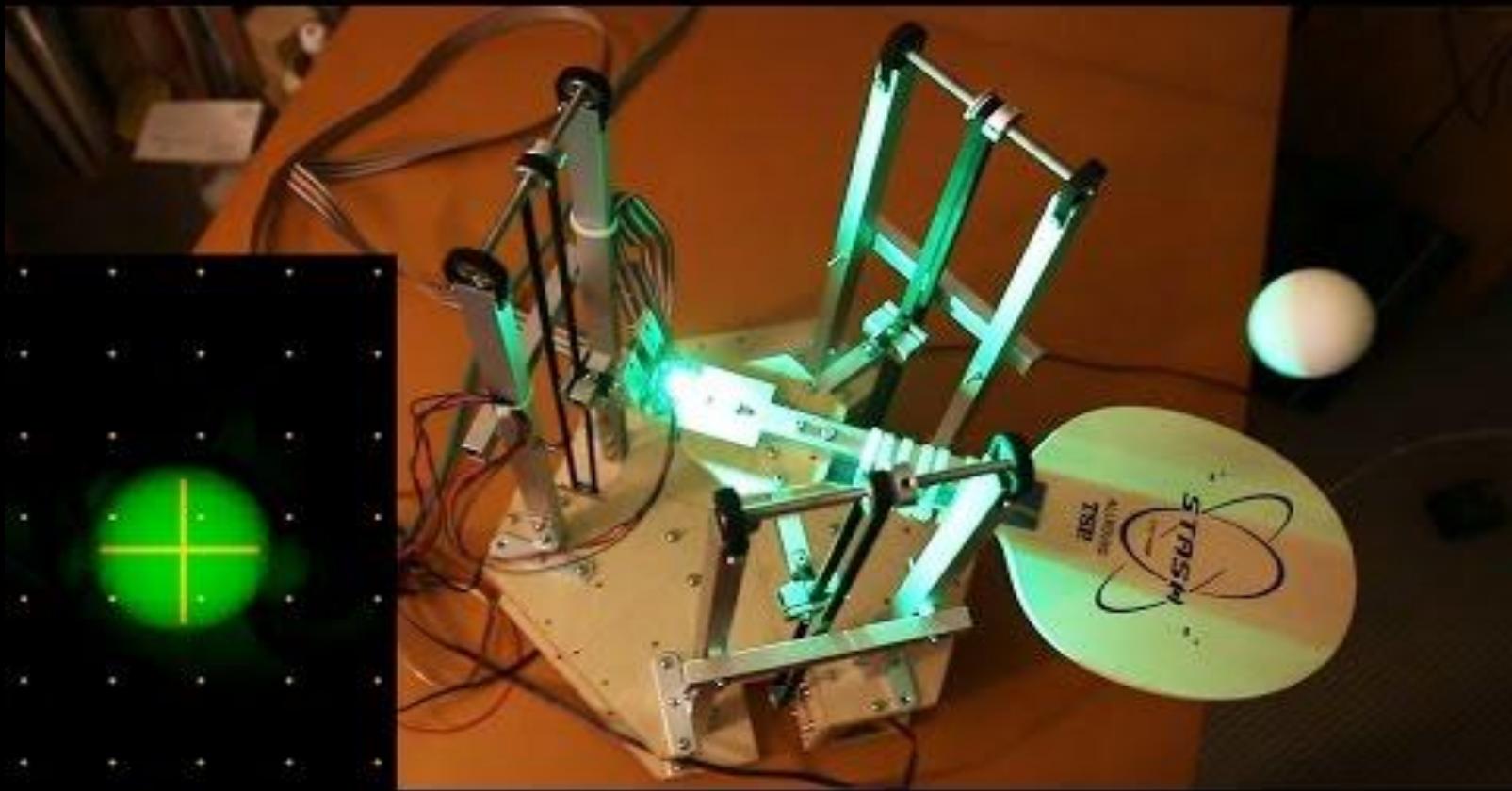
# Some inspiration



( LAPTOP FOR ILLUSTRATION )







**Want more?**  
Check the [repo](#)

# **Why do we need sensors?**

# **Understanding the basics**

## An overview of sensing techniques

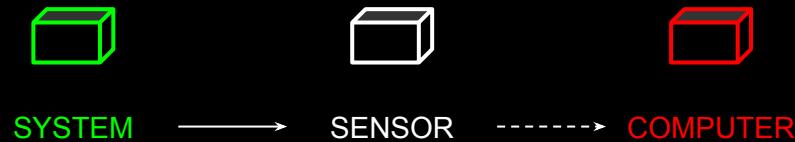


*Sensors*

# Understanding the basics - why do we need sensors?



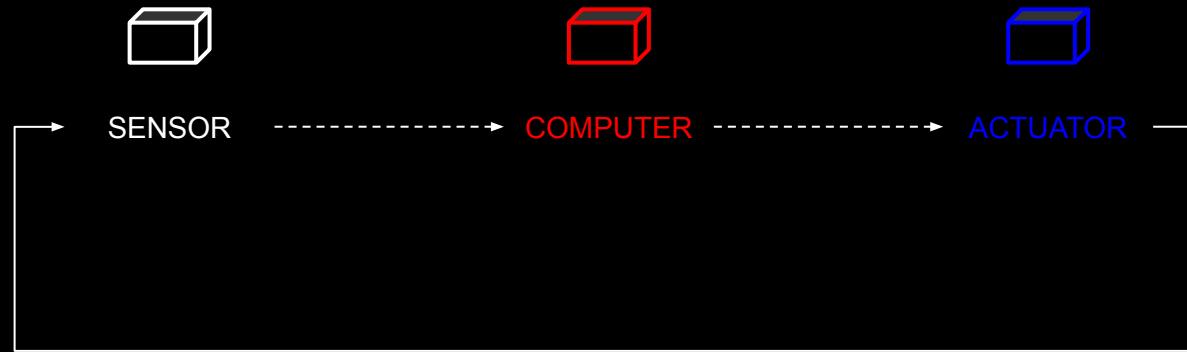
## Monitoring something



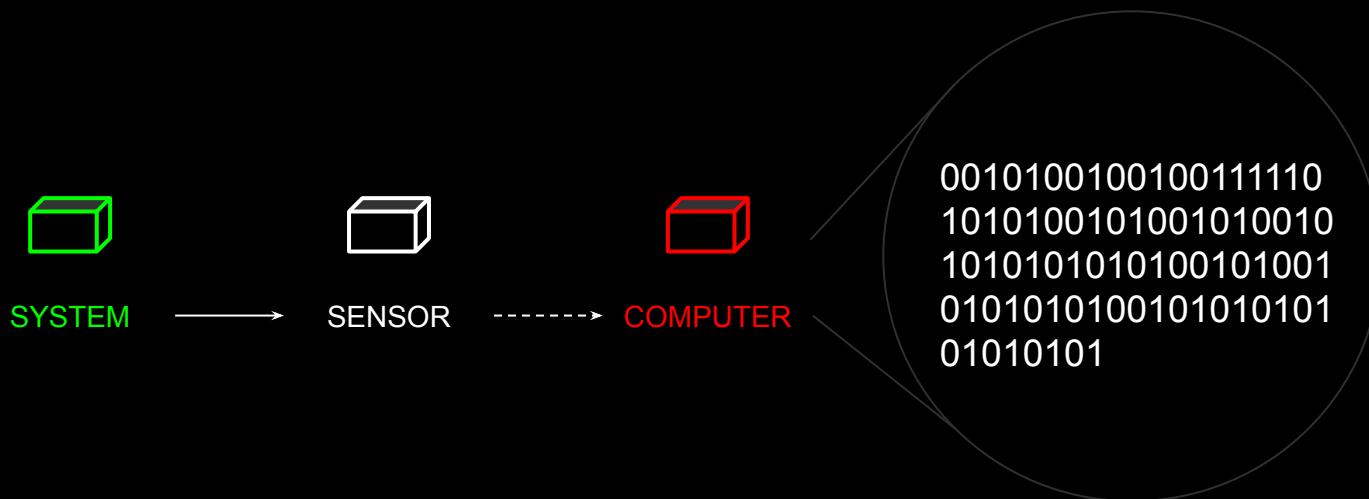
# Understanding the basics - why do we need sensors?



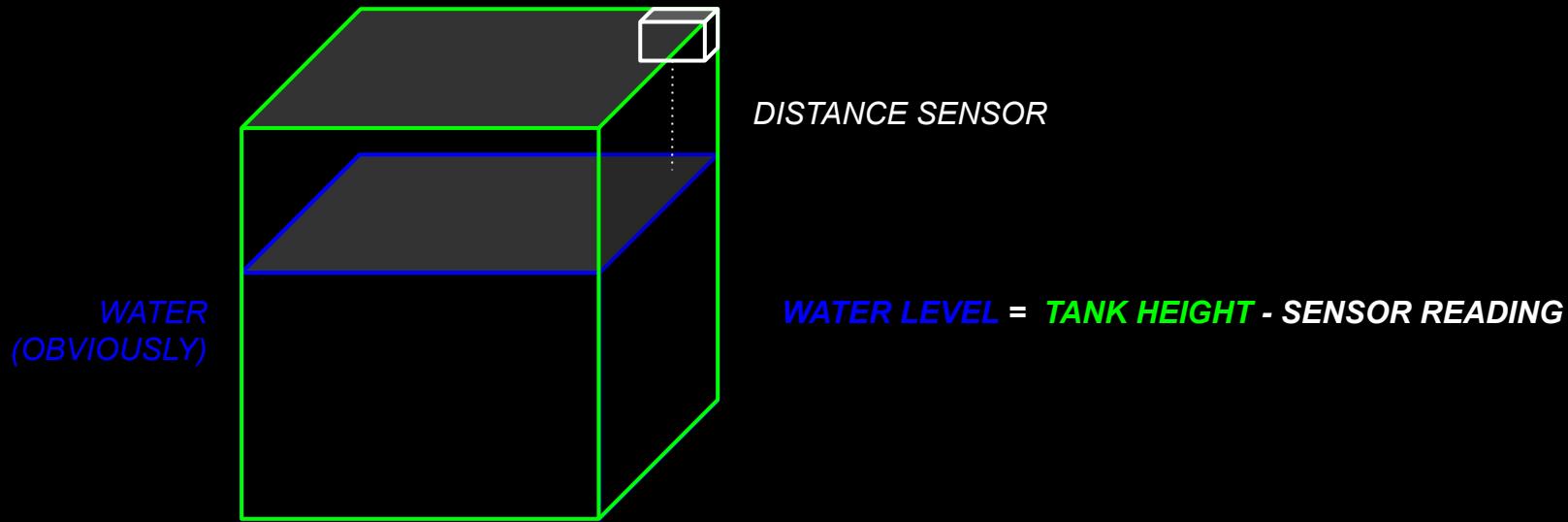
**Controlling something  
(closed loop)**



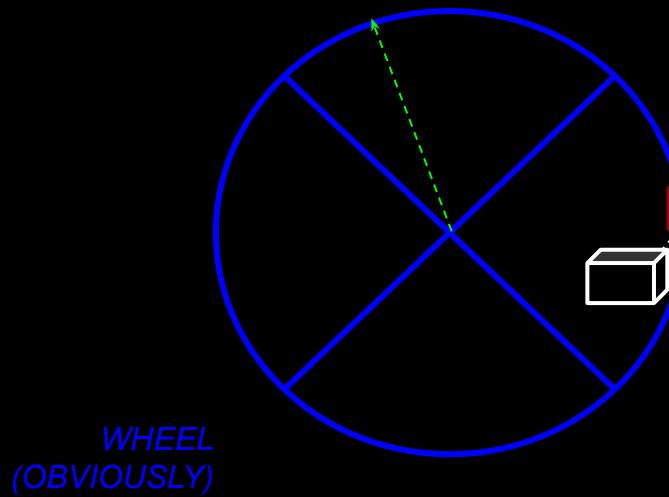
# Understanding the basics - digitalising physical phenomena



# Understanding the basics - inference



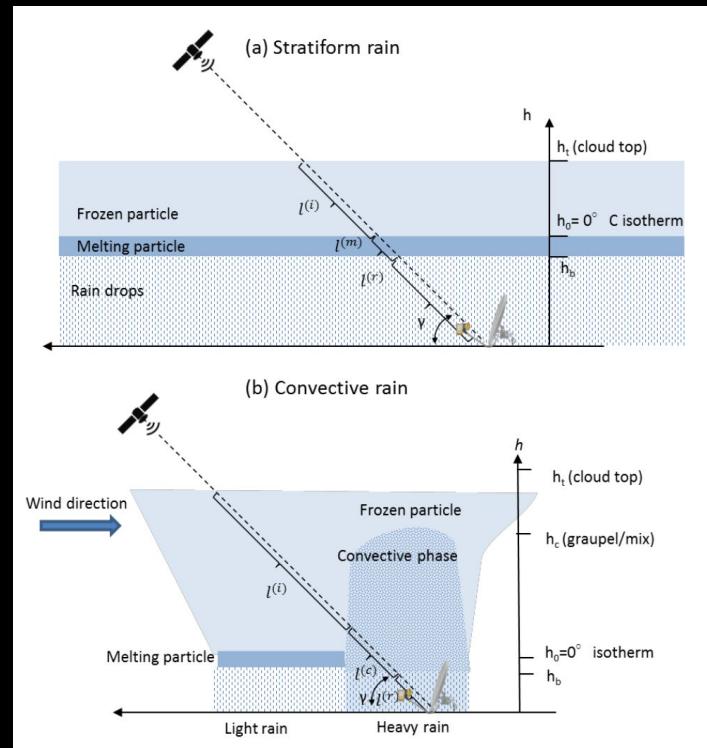
# Understanding the basics - inference



WHEEL  
(OBVIOUSLY)

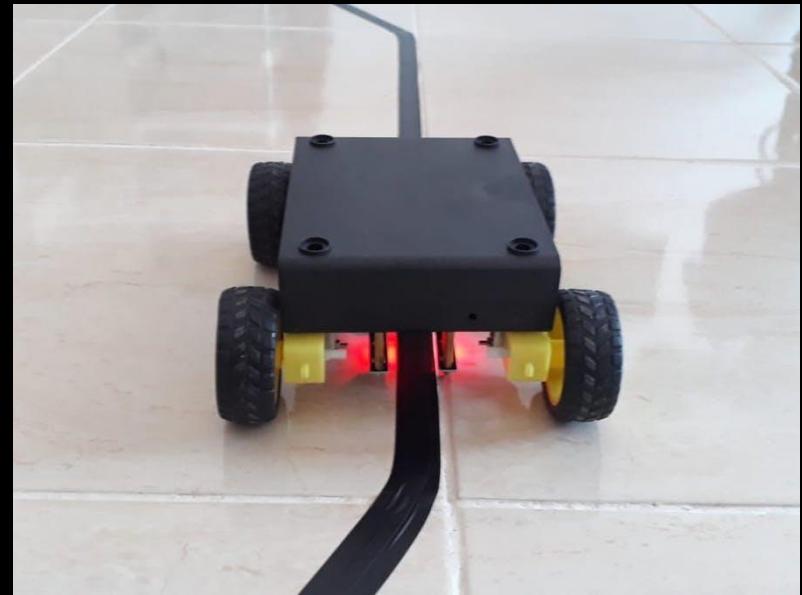
$$\text{HORIZONTAL SPEED} = (\text{PULSES}/\text{min})/4 * \text{WHEEL_RADIUS}$$

# Understanding the basics - simple vs. complex





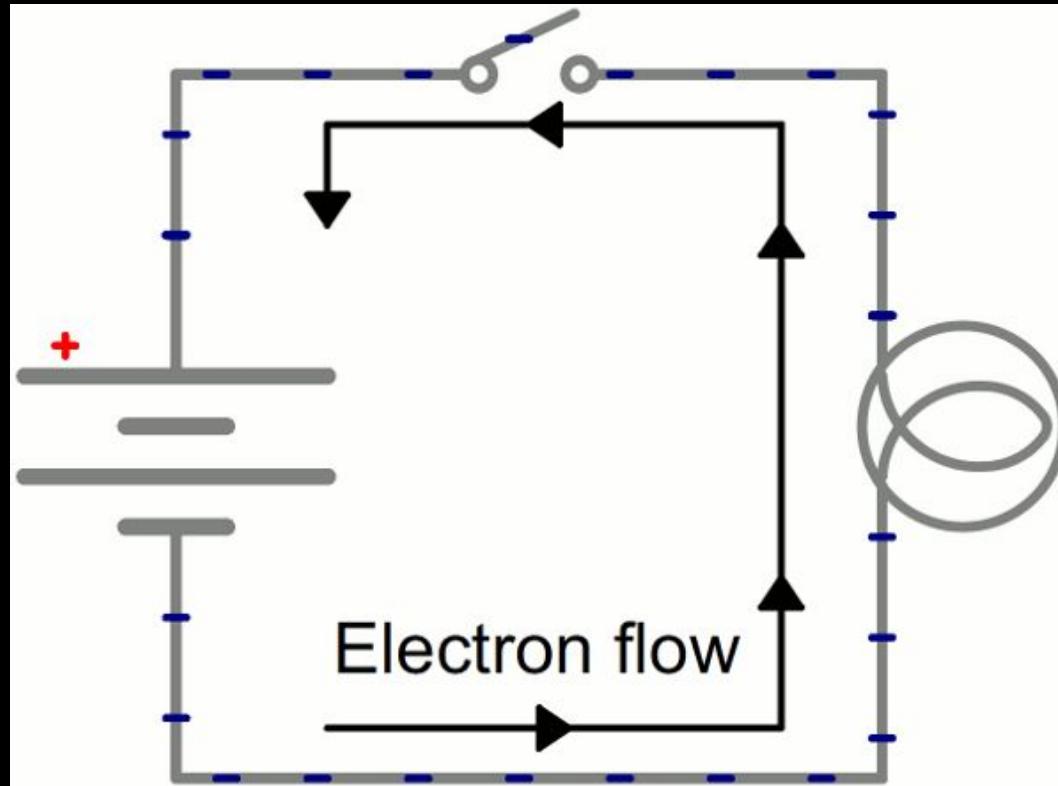
## Understanding the basics - static vs. dynamic



# Digging deeper

From signals to information

## Digging deeper - The simplest sensor



HIGH

*or*

LOW

1

*or*

0

**TRUE**

*or*

**FALSE**

**YES**

*or*

**NO**

**PERSON**

*or*

**NOT PERSON**

CAT

*or*

DOG

**5V**

*or*

**GROUND**

## Digging deeper - The simplest sensor (or maybe not-so-simple)



- *To read a digital input (**HIGH**, or **LOW**) we will use Arduino `digitalRead` function (reference <https://www.arduino.cc/reference/en/language/functions/digital-io/digitalread/>) connecting the switch to a pin in our microcontroller*
- *Depending on the type (and age) of the electronics we use the **HIGH** value corresponds to different voltages, for example: 5V, 3.3V, 1.8V...*
- *You might need to use pull-up resistors to avoid floating states*
- *You might need to debounce the signal*

## Digging deeper - The simplest sensor (or maybe not-so-simple)



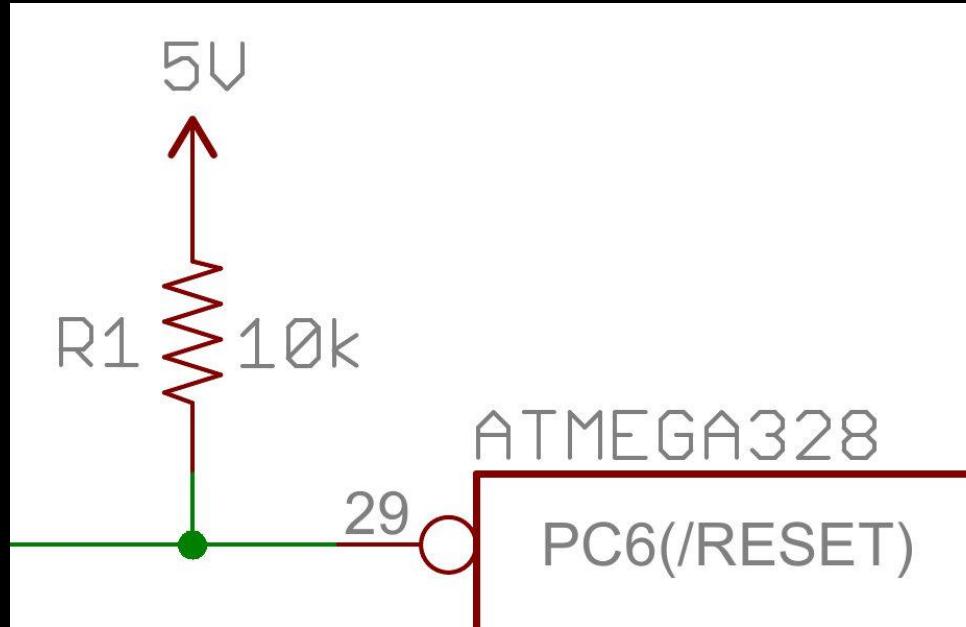
*Pull-up resistors*

*[...] pins configured as `pinMode(pin, INPUT)` with **nothing connected to them**, or with wires connected to them that are not connected to other circuits, will report seemingly **random changes in pin state**, picking up electrical noise from the environment, or capacitively coupling the state of a nearby pin.*

# Digging deeper - The simplest sensor (or maybe not-so-simple)



*Pull-up resistors*



## Digging deeper - The simplest sensor (or maybe not-so-simple)



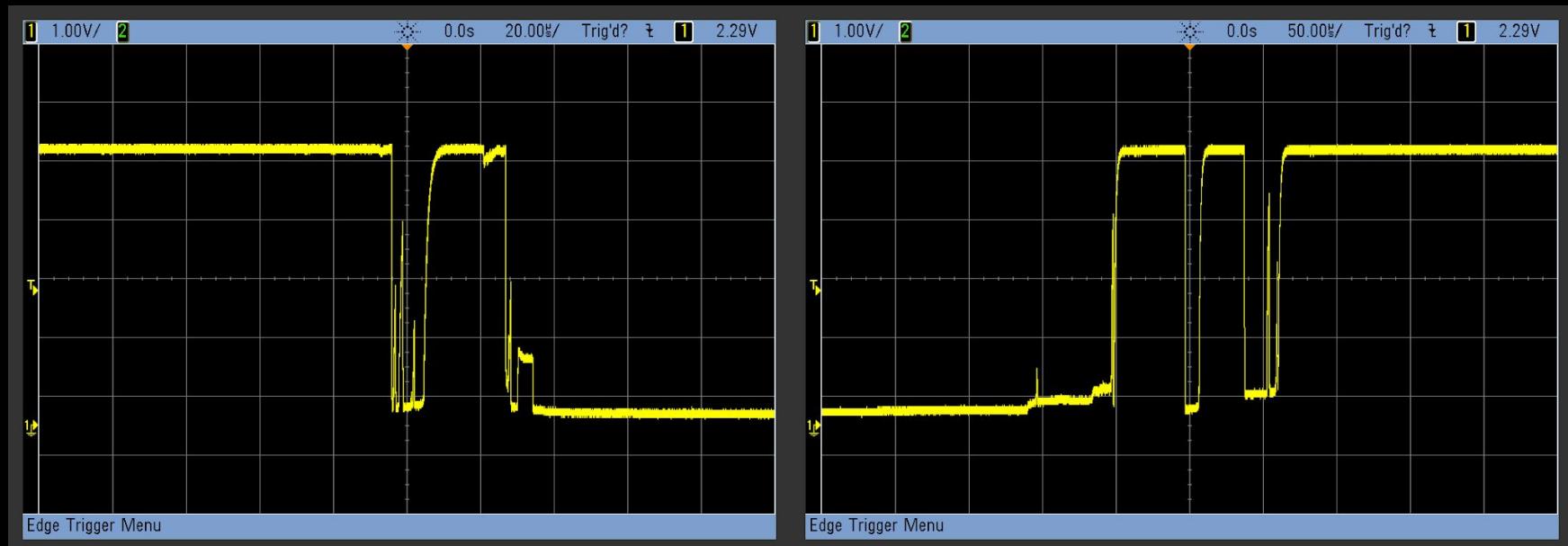
### *Debouncing*

*Pushbuttons often generate spurious open/close transitions when pressed, due to mechanical and physical issues: these transitions may be read as multiple presses in a very short time fooling the program.*

# Digging deeper - The simplest sensor (or maybe not-so-simple)



## Debouncing



# Digging deeper - The simplest sensor (or maybe not-so-simple)

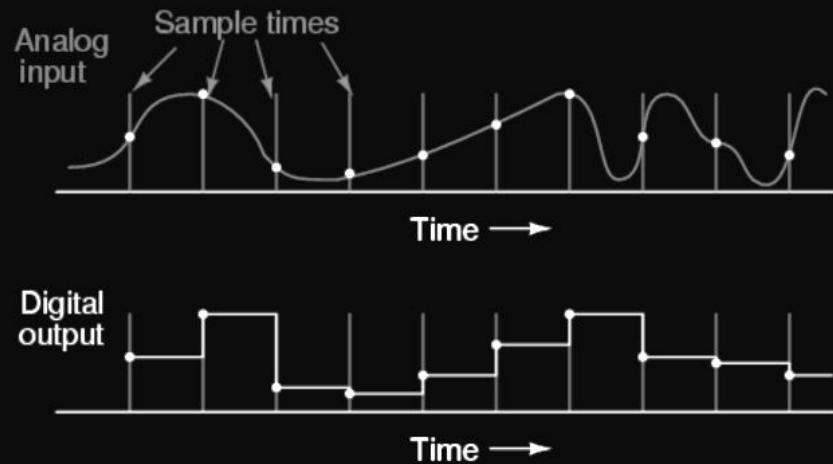


## *Reading Pulses*

*Reads a pulse (either HIGH or LOW) on a pin. For example, if value is HIGH, pulseIn () waits for the pin to go from LOW to HIGH, starts timing, then waits for the pin to go LOW and stops timing. Returns the length of the pulse in microseconds or gives up and returns 0 if no complete pulse was received within the timeout.*



## Digging deeper - Going Analog



ADC (Analog to Digital Converter) 10 bits

## Digging deeper - Going Analog



*In an **ADC** circuit there are several steps to go from an **analog signal to a digital value**. The first stage is called **Sampling and Holding**:*

An analog signal continuously changes with time, in order to measure the signal we have to keep it steady for a short duration so that it can be sampled.

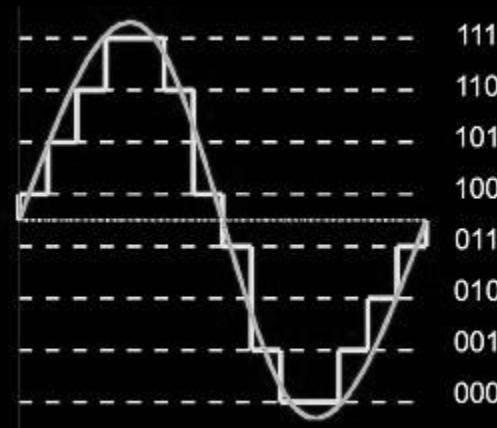
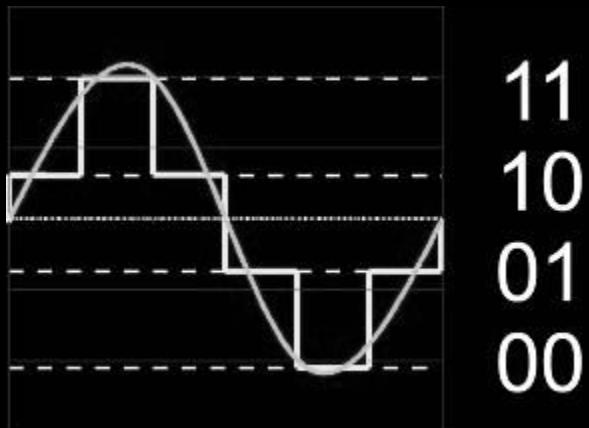
*The next stage is **Quantizing and Encoding**:*

On the output of (S/H), a certain voltage level is present. We assign a numerical value to it. The nearest value, in correspondence with the amplitude of sampling and holding signal, is searched.

## Digging deeper - Going Analog



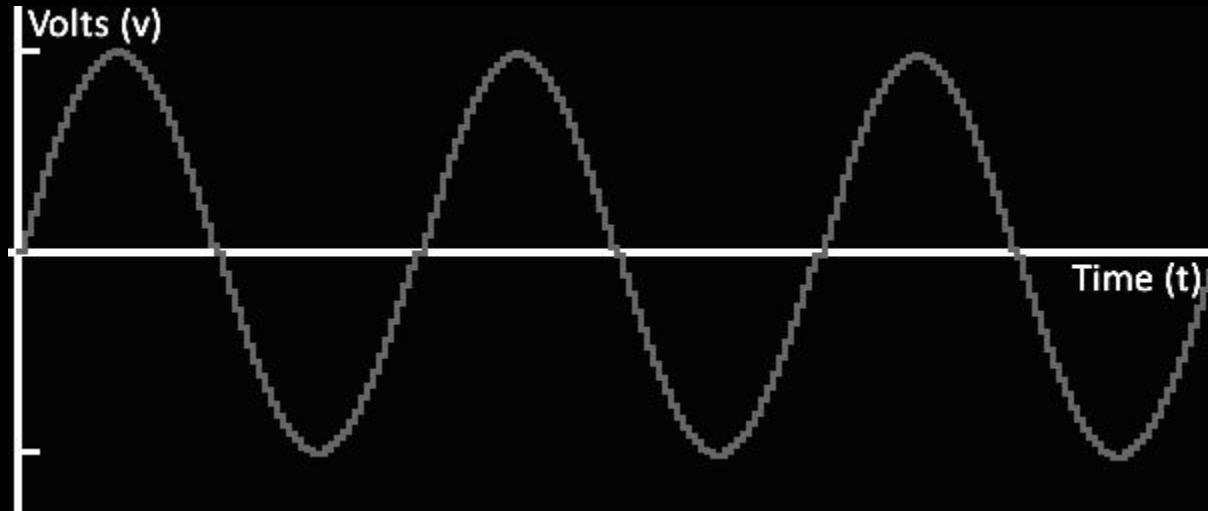
The **resolution** of an ADC is measured in bits, a one bit resolution ADC is capable of delivering  $2^1$  different values (0 and 1). The Arduino UNO has a **10 bit** integrated ADC that means it can deliver  $2^{10}$  values **from 0 to 1023**.





## Digging deeper - Going Analog

The **resolution** of an ADC is measured in bits, a one bit resolution ADC is capable of delivering  $2^1$  different values (0 and 1). The Arduino UNO has a **10 bit** integrated ADC that means it can deliver  $2^{10}$  values **from 0 to 1023**.

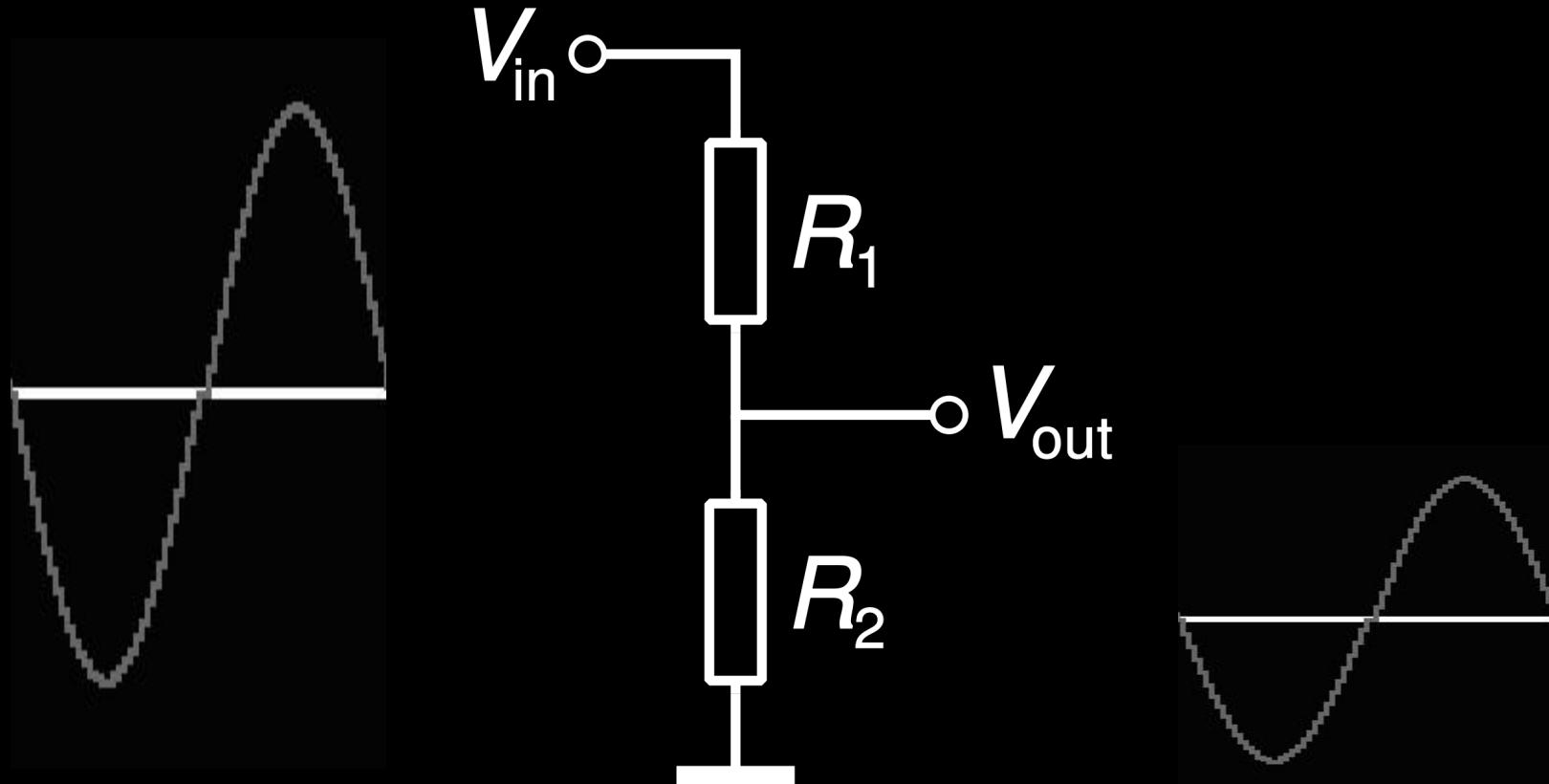




## Digging deeper - Going Analog

- *To read an analog input (**continuous values**) we will use Arduino `analogRead` function (reference <https://www.arduino.cc/reference/en/language/functions/analog-io/analogread/>) connecting the sensor to a pin in our microcontroller (**which needs to have an ADC**)*
- *Depending on the sensor type, we might need more or less **resolution***
- *You might need to use a voltage divider **if the sensor range is too large with respect to that of the ADC***

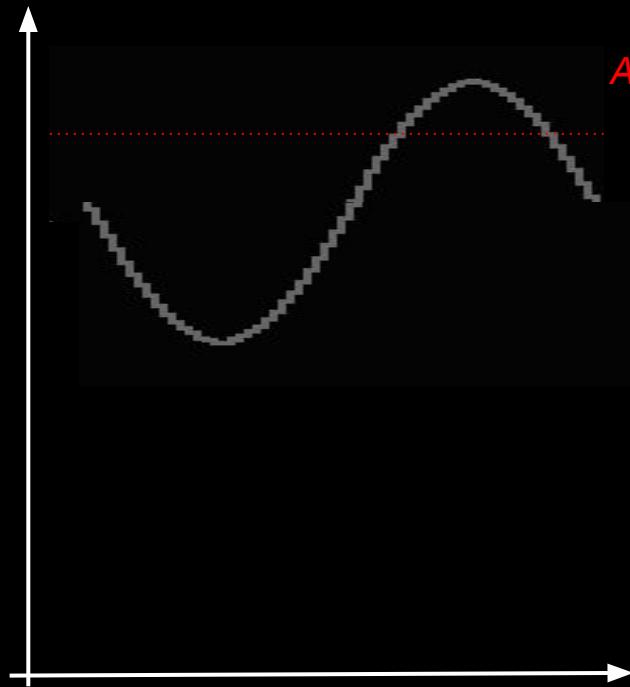
## Digging deeper - Going Analog / Voltage dividers



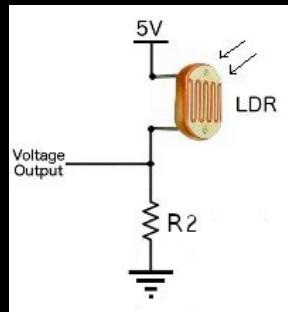


## Digging deeper - Going Analog / Voltage dividers

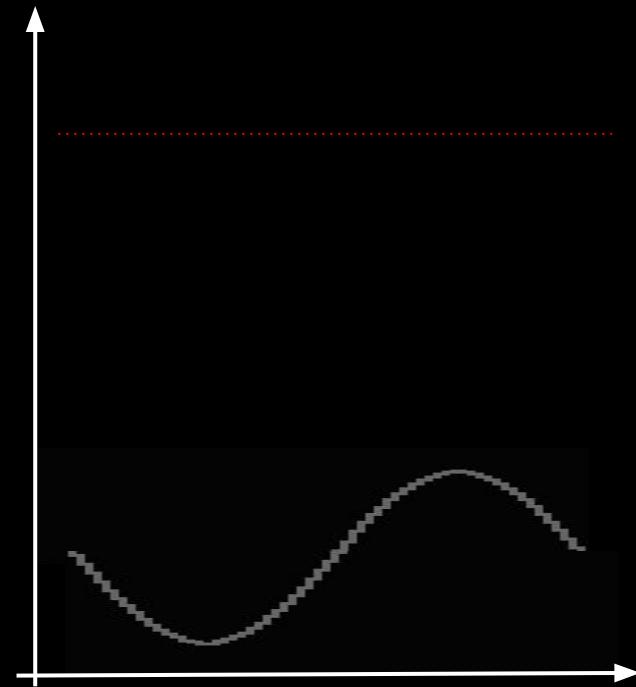
*WITHOUT* Voltage Divider



*ADC range*



*WITH* Voltage Divider



## Digging deeper - Analog and digital / Understanding voltage levels



- *Sometimes the sensors we use have a different voltage range and for that reason we use a voltage divider to adapt the signal to the range of our microcontroller*
- *We can also use a voltage divider to measure a sensor that is not **centered** with our ADC range*



# Example time!



**Before we continue...**  
I'm sorry



# Digital sensors

# Digital Sensors



- Some sensors have more complex technical specifications and might even have a tiny microcontroller
- We can make our microcontroller (Arduino, Raspberry Pi, ...) talk with those sensors by using digital communication protocols such as I2C, SPI, ...





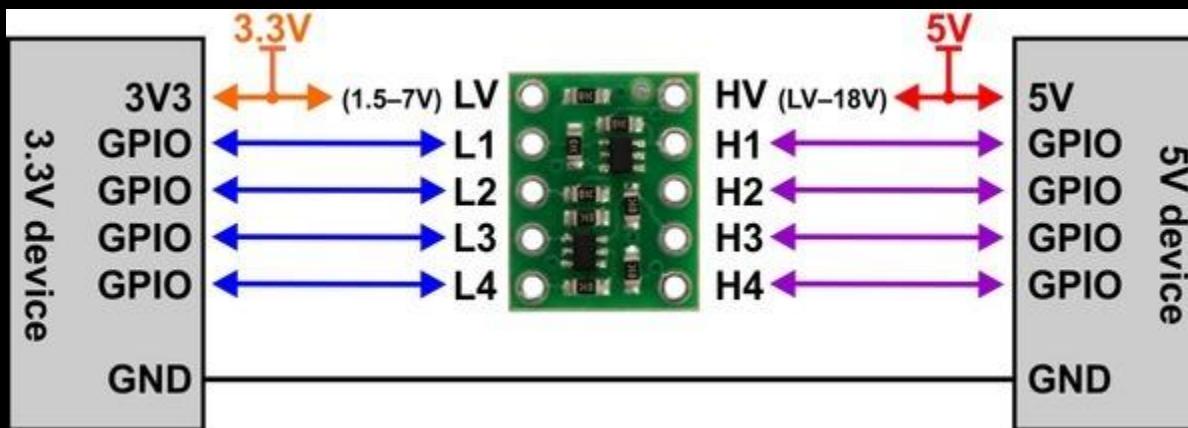
# Digital Sensors

- *To talk with a digital sensor we will mostly use Arduino Wire (reference <https://www.arduino.cc/en/Reference/Wire>) or SPI (<https://www.arduino.cc/en/Reference/SPI>)*
- *If using a raspberry pi, you will need to enable I2C interface in raspi-config*
- *Part of the processing (heavy lifting) is done by the sensor itself*
- *You might need to use a level shifter to adapt voltage levels*



## Digital Sensors / Level Shifter

As digital devices get smaller and faster, once ubiquitous 5 V logic has given way to ever lower-voltage standards like 3.3 V, 2.5 V, and even 1.8 V, leading to an ecosystem of components that need a little help talking to each other. For example, a 5 V part might fail to read a 3.3 V signal as high, and a 3.3 V part might be damaged by a 5 V signal.





# Example time!



# How to choose a sensor

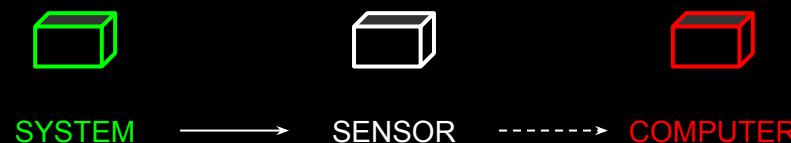
Main aspects to consider

# How to choose a sensor - Who does the heavy lifting?

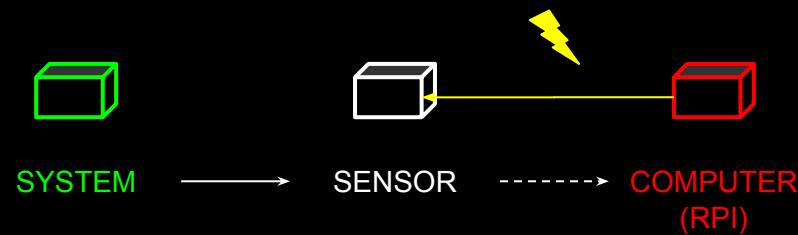
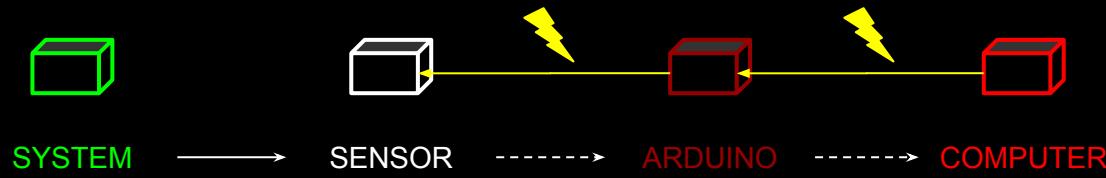


*First - What is heavy lifting?*

1. *Energy consumption - who provides it?*
2. *Computing power*
3. *Computing power (post) and because of that, data storage*



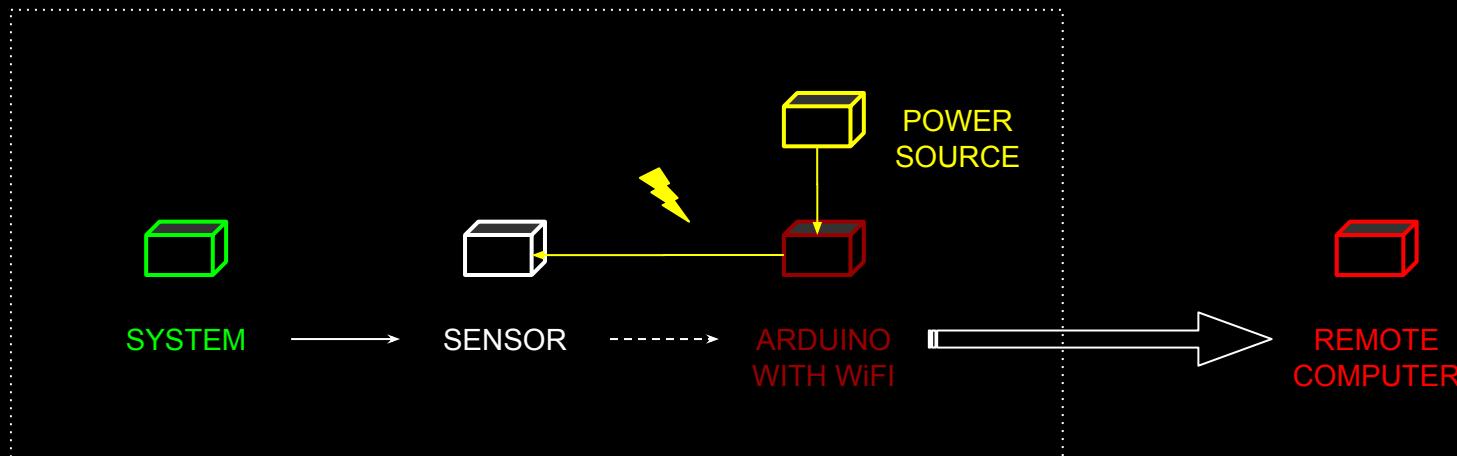
# How to choose a sensor - Who does the heavy lifting?



# How to choose a sensor - Who does the heavy lifting?



MOVING ROBOT



## How to choose a sensor - What are the voltage levels?



*Remember, this can kill your sensor*

1. *Are both the same? Good!*
2. *Are they different? Can it be solved with a voltage divider (analog or digital input) or a level shifter (digital sensor)?*



# The three rules of every project

**1. *Do not overkill - the simpler, the better***

**2. *Always look for existing solutions first***

- *look for well documented sensors [learn.sparkfun.com](https://learn.sparkfun.com), [learn.adafruit.com](https://learn.adafruit.com), [github.com](https://github.com)...*

**3. *Do not forget about software - use libraries, specially for digital sensors***



# Cameras

Some basics



# Cameras - Some basics

**We have three types of cameras**

1. *RGB cameras - [Mobius action camera](#)*
2. *Depth cameras (normally with RGB, so RGBD) - [Kinect v1](#) and [INTEL REALSENSE](#)*
3. *Thermal cameras - [Flir one](#)*

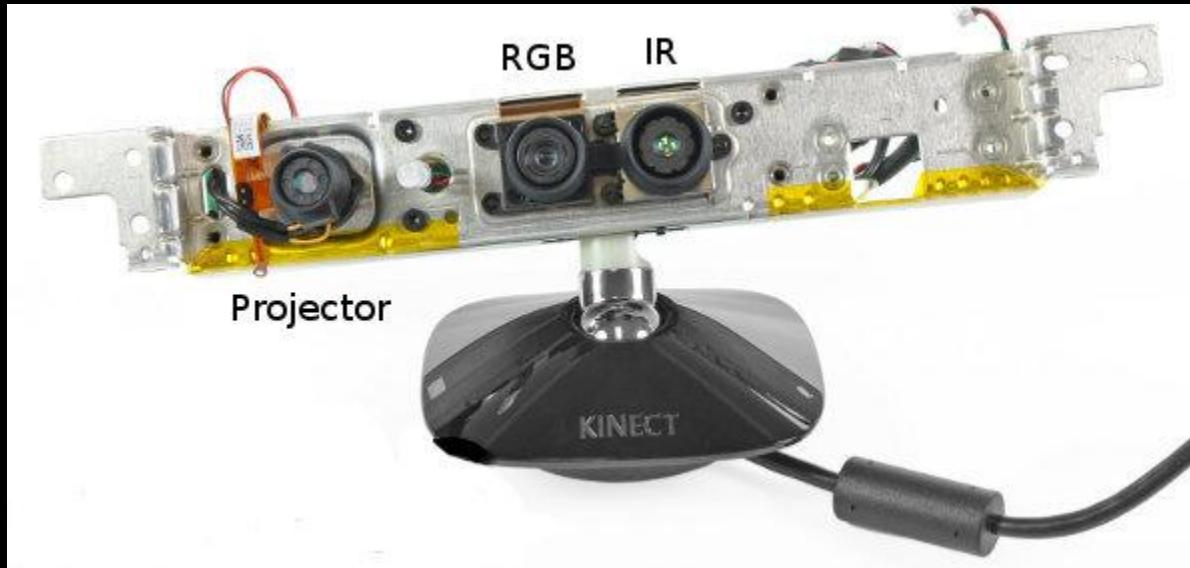
**Things to consider:**

1. *RGBD or RGB cameras for positioning are both valid, one putting more emphasis on the post-processing stage with more calibration needs*
2. *Make sure you have enough processing power for image capture and/or storage*



# Cameras - Some basics

## *RGBD cameras*

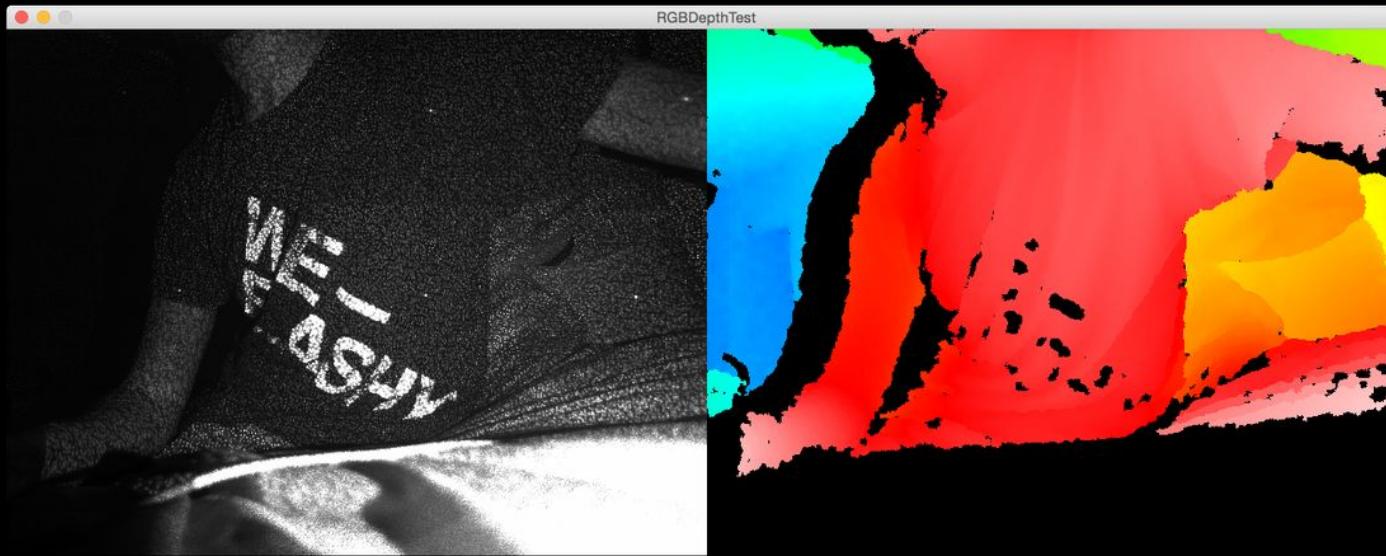


*Image found found at: [https://wiki.ros.org/kinect\\_calibration/technical](https://wiki.ros.org/kinect_calibration/technical) originally from <http://www.ifixit.com/Teardown/Microsoft-Kinect-Teardown/40661>*



# Cameras - Some basics

## RGBD cameras

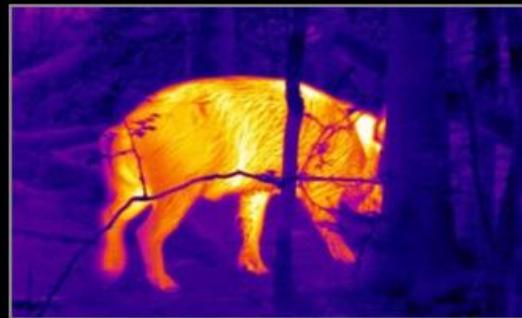


Press 'd' to enable/disable depth Press 'r' to enable/disable rgb image Press 't' to enable/disable IR image Press 'c' to enable/disable color depth image UP and DOWN to tilt camera Framerate: 60

# Cameras - Some basics



## *Thermal imaging cameras*



*Located far from the visible light spectrum thermal imaging doesn't require a light source at all. Instead, thermal imaging works by collecting the infrared radiation emitted by all objects with a temperature above absolute zero. Differences in radiation between objects make up the image which is then shown on screen.*

**Well done!**