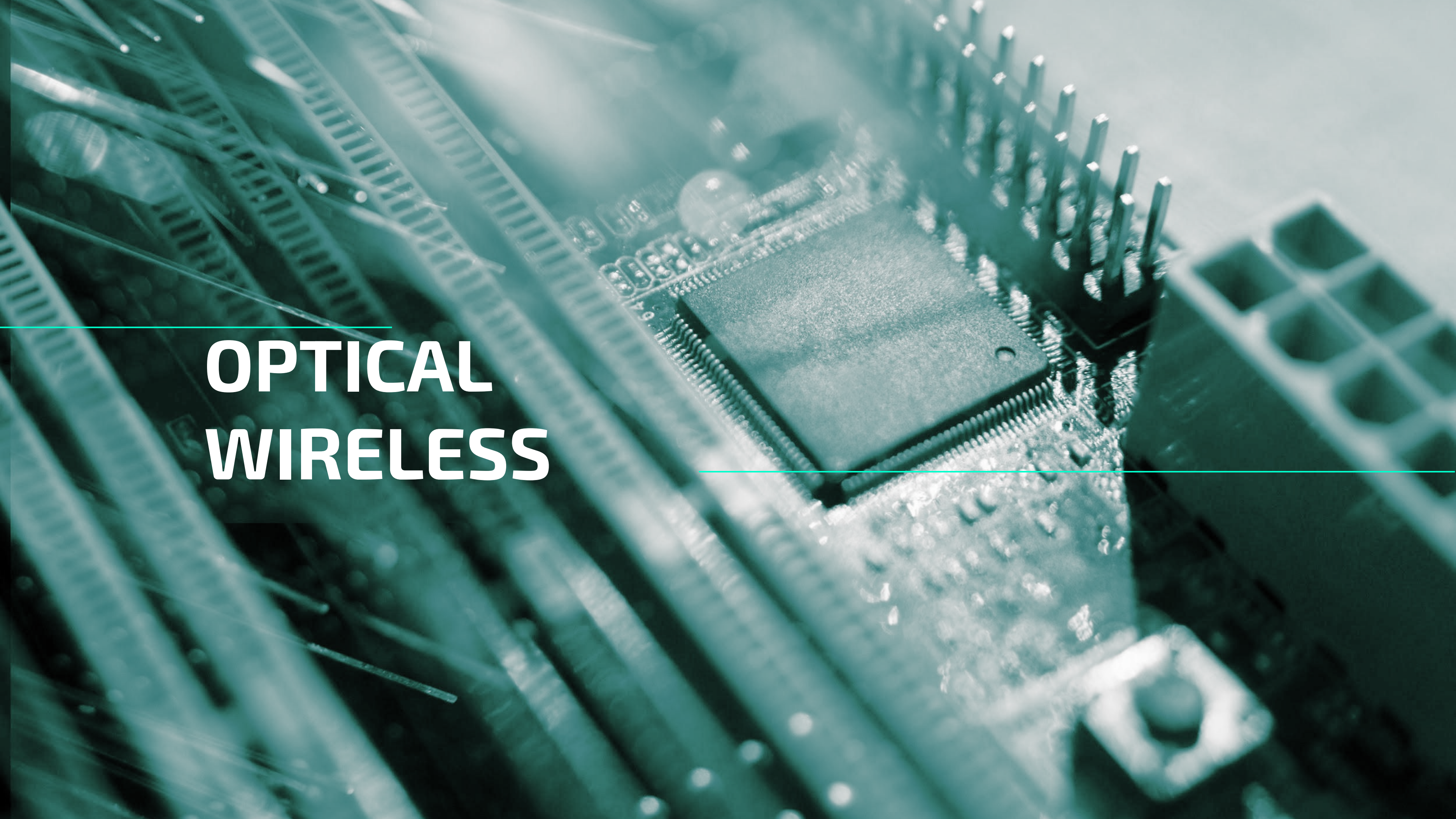
OPTICAL WIRELESS



CHANNEL CODING FOR VISIBLE LIGHT COMMUNICATIONS

Smart Environments - Spring Semester

Francesco Russo, Michele Cernigliaro

Github repo: https://github.com/francescoweb1010/OpticalWirelessSystem

TABLE OF CONTENTS

01

VISIBLE LIGHT COMMUNICATION

Data through leds

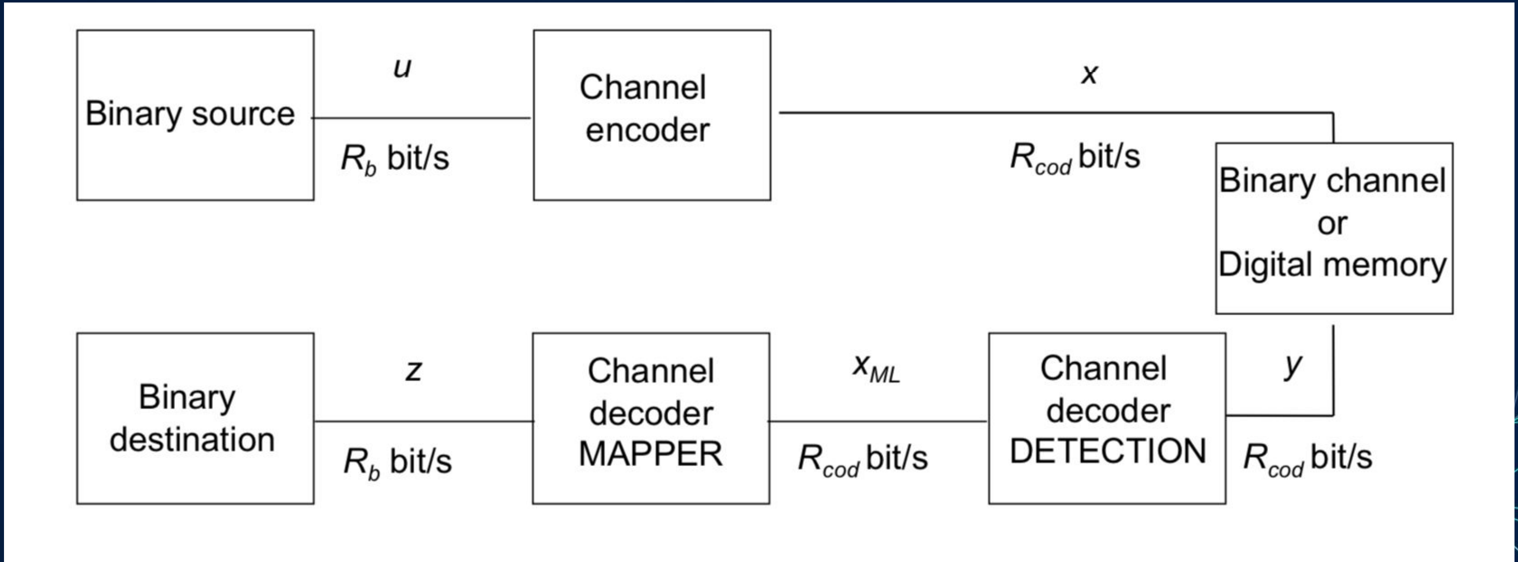
01. Visible light communications
The framework

- Implementation (Hardware and Software)
  - Visible light communication between IoT devices

- Applications
  - Communications of Smart Devices
  - Military applications
  - Underwater applications
  - Industrial applications
  - Healthcare applications

- Send and receive (binary) string using:
  - Led
  - Photodiode

01. Visible light communications
The framework



Generic binary transmission with block coding
Prof. Mauro Biagi, Smart Environments 2020 class notes

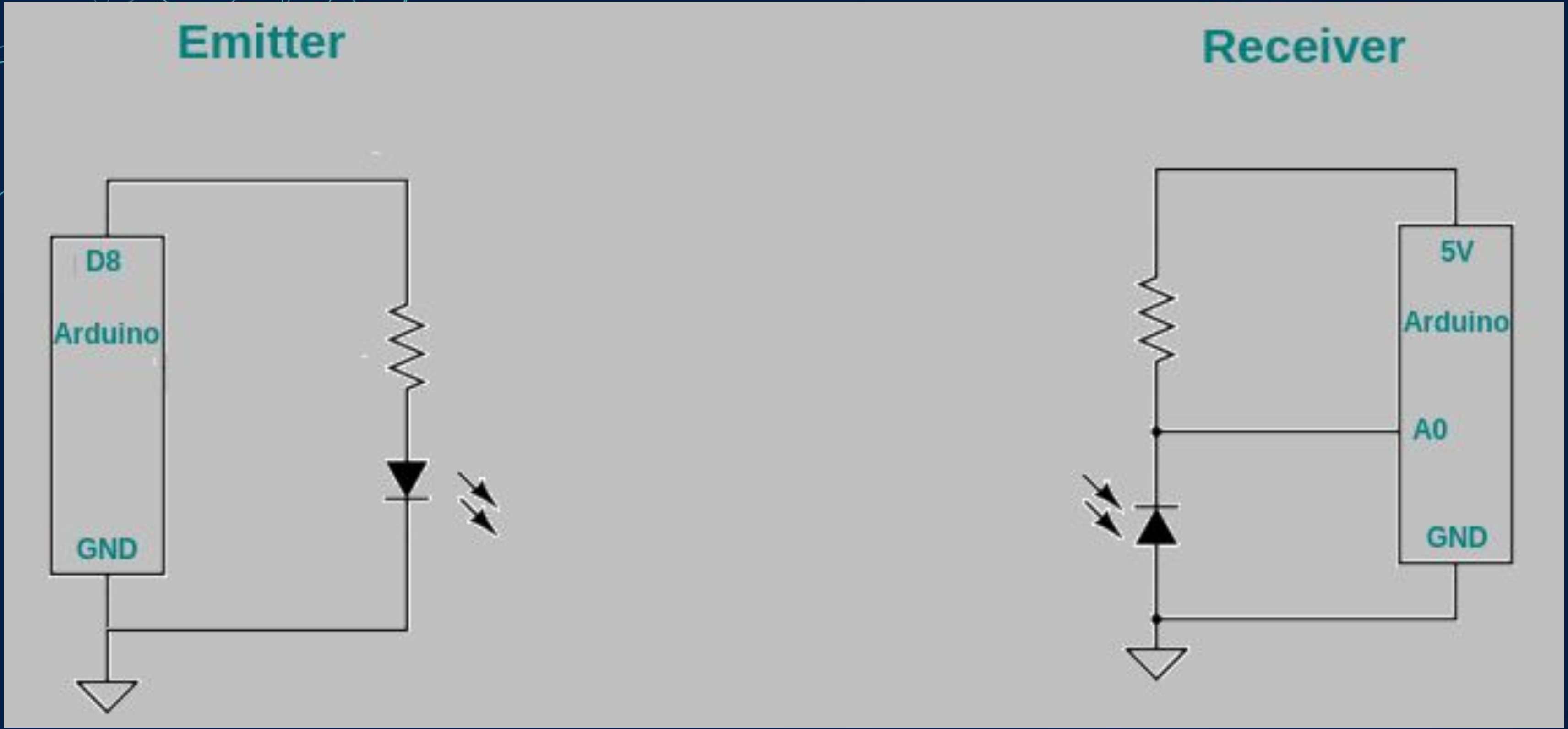
02

IMPLEMENTATION

02 - IMPLEMENTATION

- 2 arduino boards
  - 1 acting as a sender
    - Using a circuit with a led
  - 1 acting as a receiver
    - Using a circuit with a photodiode
- Arduino One (ATmega328) and Micro (ATmega32u4) (different time resolutions)

02 - IMPLEMENTATION



02 - IMPLEMENTATION

- Programming languages:
    - Arduino (for microcontrollers internal sketches)
    - Python (for the message reception)

- Serial port control software (minicom, miniterm, pyserial, putty)

- Real time message reception algorithm in Python
    - easily portable to devices such as Raspberry PI

03

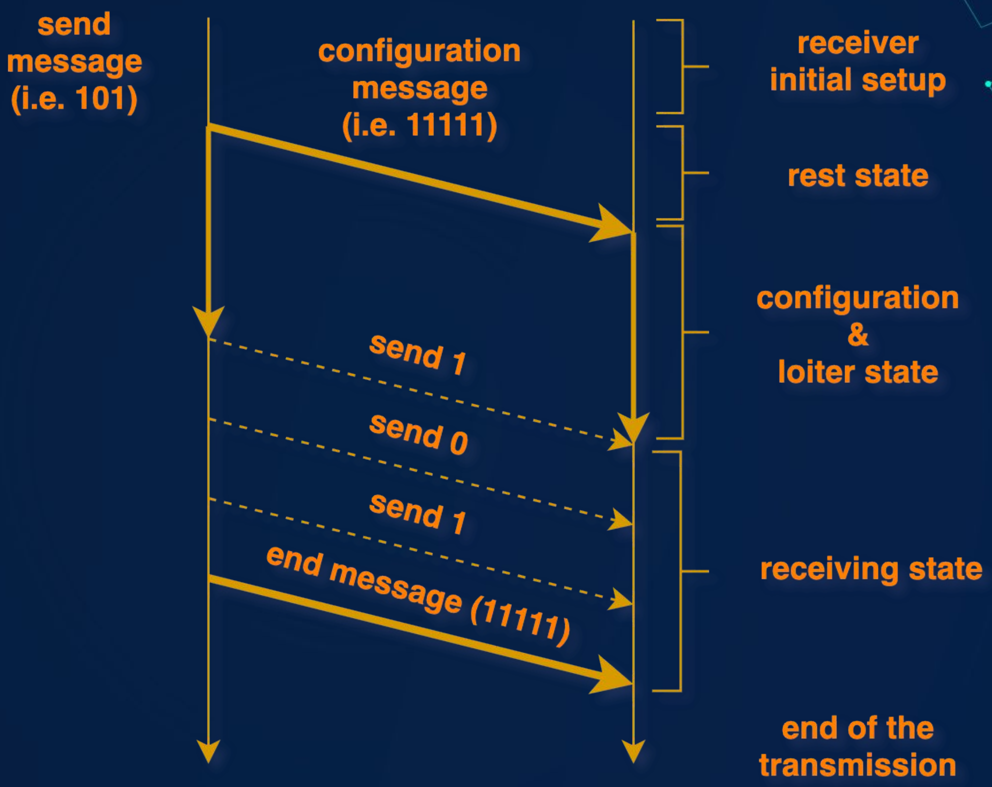
MESSAGE TRANSMISSION

Channel coding, sending encoded message

```
int i = 0, b = 0;
int sending_flag = 0;
char source[] = "11110000111100001111000011110000111100001111";

void setup() {
  pinMode(8, OUTPUT);
  Serial.begin(9600);
}

void loop() {
  if (Serial.available() > 0){
    b = Serial.read();

    if (b == 49){
      digitalWrite(8, HIGH);
      delay(2000);
      digitalWrite(8, LOW);
      delay(2000);

      sending_flag = 1;
    }
  }
```

```
  if (sending_flag == 1){
    for(i = 0; source[i] != '\0'; i++){
      Serial.println(source[i]);
      if (source[i] == '1') {
        digitalWrite(8, HIGH);
      }
      else {
        digitalWrite(8, LOW);
      }
      delay(200);
    }
    sending_flag=0;
    digitalWrite(8, HIGH);
    delay(2000);
    digitalWrite(8, LOW);
  }
}
```

03 - Message transmission



03 - Message transmission

- Sending the message with arduino is straightforward
  - Given a binary string (1010101)
    - If the value is 1
      - Led on for i.e. 200 ms
    - Else
      - Led off for i.e. 200 ms
  - This is done within each message
    - Configuration message
    - Message
    - End message

03 - Message transmission
Channel coding

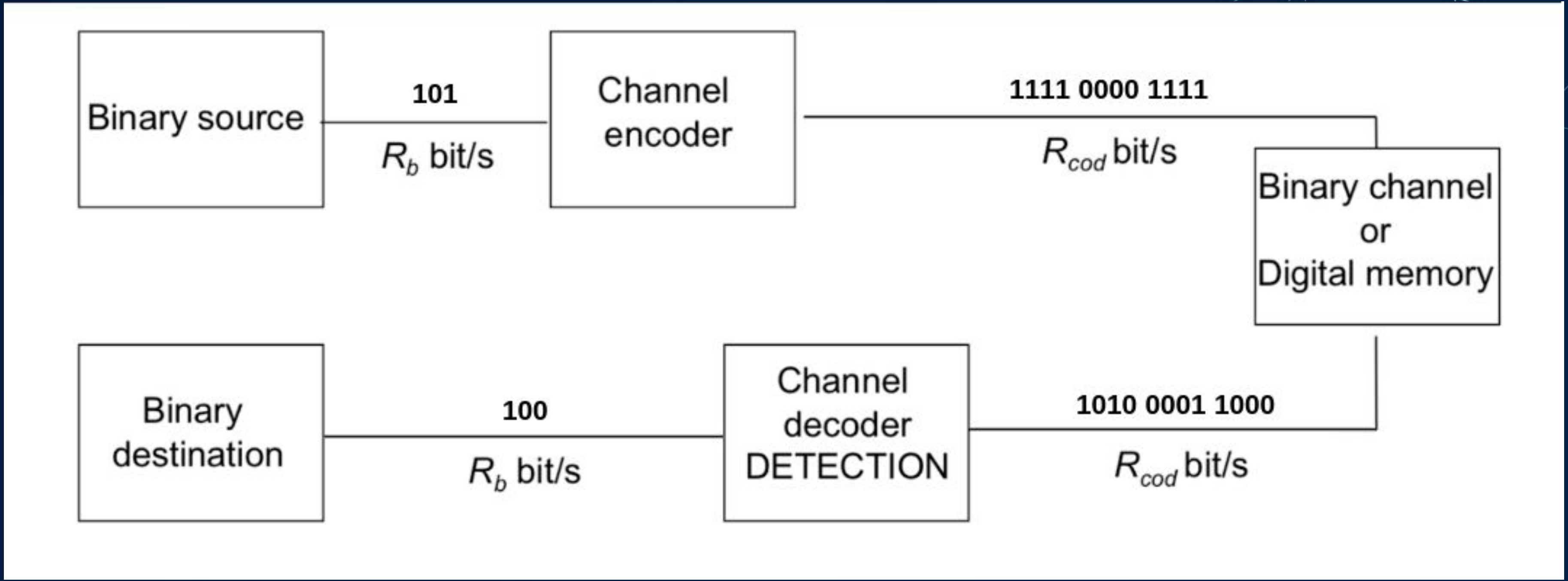
- Channel coding in order to reduce interference, poor signal, noise

- We performed a simple encoding/decoding by repeating *k* times each dataword (repetition code)

- Encoding (k=4)
  - 1 ⇒ 1111
  - 0 ⇒ 0000

- Decoding (k=4) - Decoding by majority decision
  - 1111 ⇒ 1
  - 0000 ⇒ 0

03 - Message transmission
Repetition code - example



04

MESSAGE RECEPTION

Structure of the reception implementation

```
float sensorValue = 0;

void setup() {
  pinMode(A0, INPUT);
  Serial.begin(9600);
}

void loop() {
  sensorValue = 1024 - analogRead(A0);
  Serial.print(millis());
  Serial.print("; ");
  Serial.println(sensorValue);
}
```

04 - Receiver states

Rest state

The device prepares for being able to reach a configuration message

Configuration state

Configuration message occurs, the device analyze the first message to set thresholds

Loiter state

Waiting time for both devices before sending and receiving the message

Receiving state

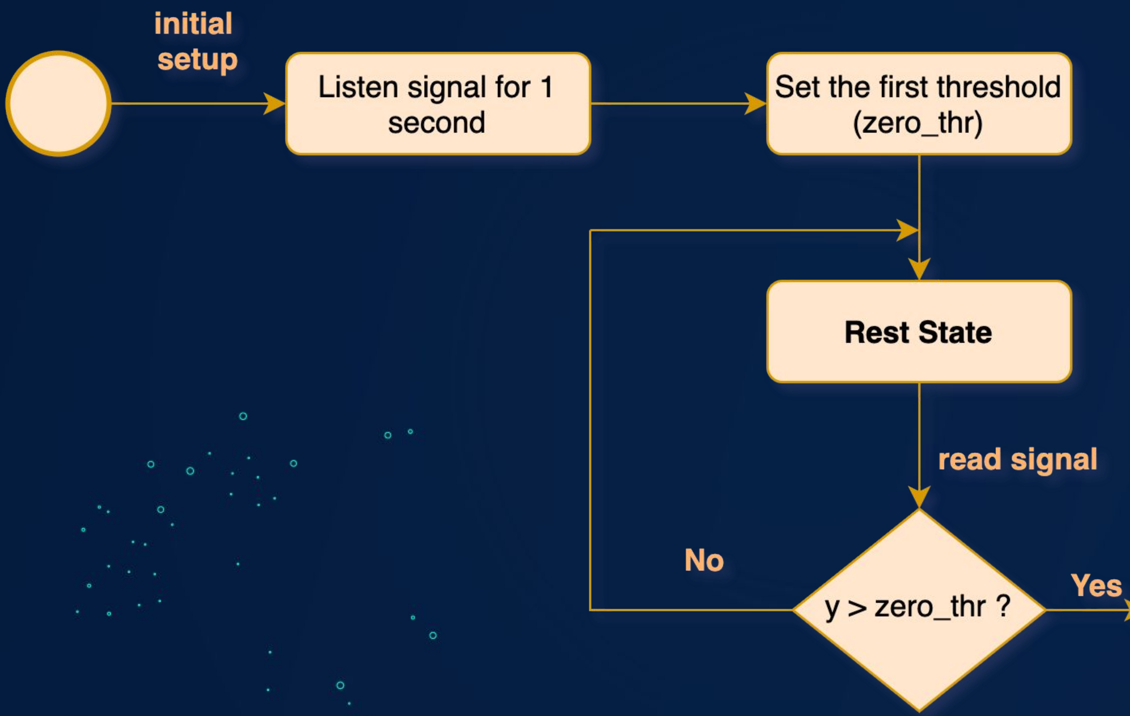
Receiving phase, the receiver analyze in real time the signal.

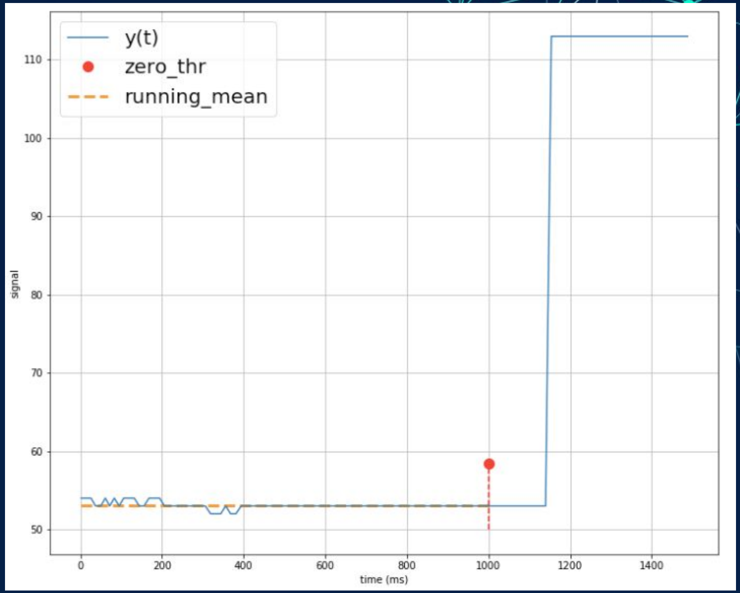
Conclusion state

The receiver elaborates the final message

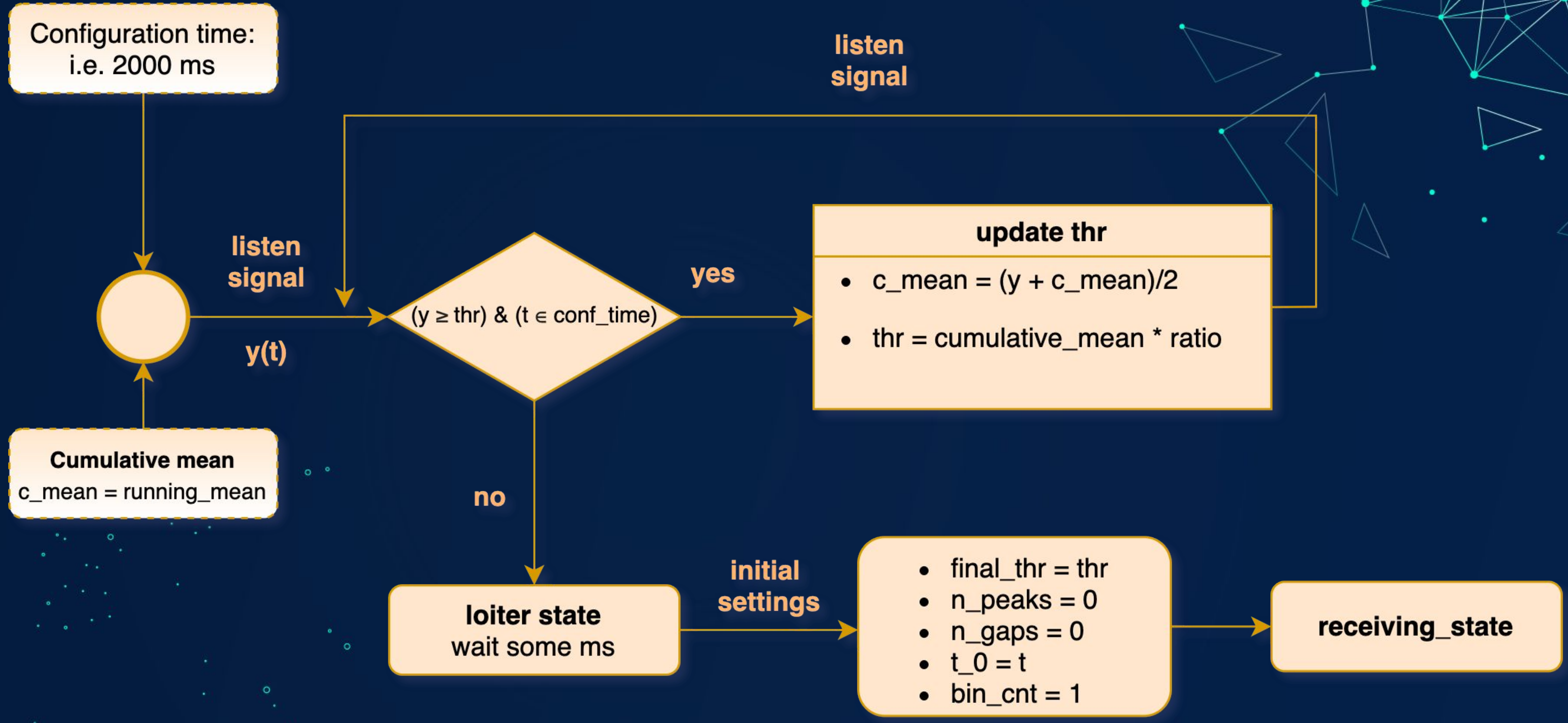
From codeword to dataword

04 - Rest State - Configuration state transitions





04 - Configuration ⇨ Synchronization ⇨ Receiving state

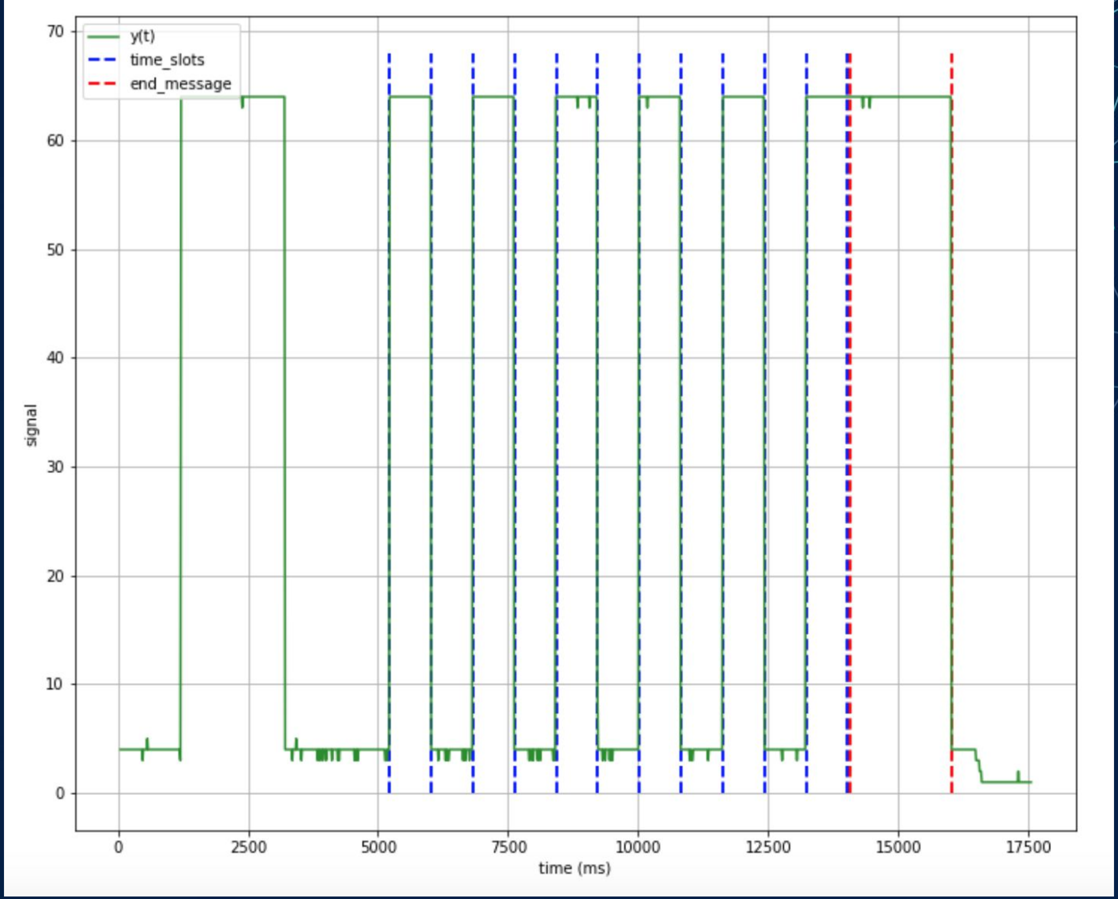


04 - Receiving state

- The main idea:
  - Divide the message in time slots
  - Count peaks and gaps occurrences
  - Assign for each interval the number w.r.t. the most frequent signal

Peaks are associated to 1 (led on)
Gaps are associated to 0 (led off)

Notice: this is done <u>real time</u> !!
that's why we need
different flags such as a bin counter



04 - Receiving state

```
if receiving_state:
    if y >= thr:
        n_peaks += 1
    else:
        n_gaps += 1
    if t >= t_0 + cnt*signal_delta:
        cnt += 1
        if n_peaks >= n_gaps:
            message = message + "1"
        else:
            if message[-10:] == "1111111111" or message[-10:] == "0000000000":
                message = message[:-10]
                receiving_state = False
                conclusion_state = True
                break
            message = message + "0"
        n_peaks = 0
        n_gaps = 0
```

05

ANALYSIS

Stress test using different configurations

05 - Analysis

- Different configurations:
  - One string example:
    - 10101010101
  - We used different configuration to “stress” our algorithm implementation
    - Time_delta : 50, 100, 150, 200 ms (20, 10, 6.67, 5 bit/s)
    - Distance: 10, 30, 50 cm
    - House Light: on, off
- Metric
  - After the encoded message reception
    - Message Decoding
    - Levenshtein distance between original and received message

05 - Results

- We run 34 correct experiments
- Almost half of the times (47%) the message has been sent correctly (LD=0)
- Surprisingly, distance and light didn’t affect the result as much as the bitrate
- Only messages with time delta of 150 and 200 ms were able to be sent correctly

Correctly sent results

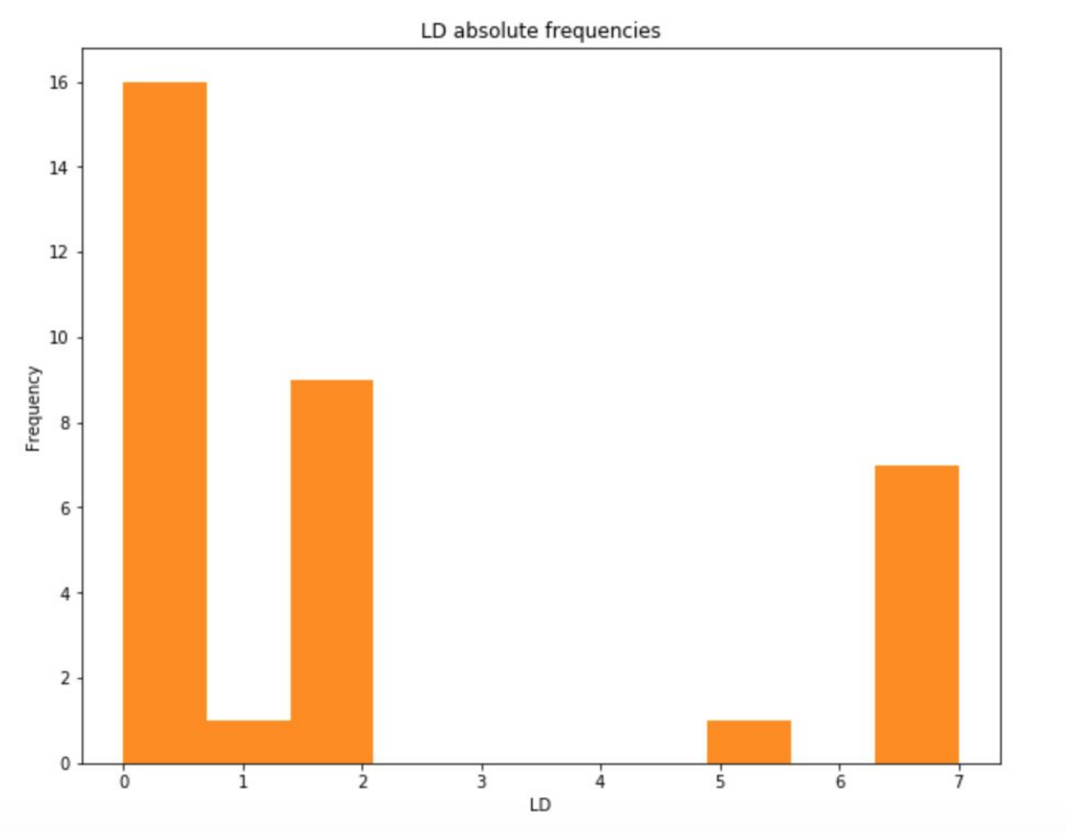
	file	model	cm	ms	light	message	final_message	LD
0	opt_s30_t200_d.log	micro	30	200	False	11110000111100001111000011110000111100001111	10101010101	0
3	opt_s10_t200.log	micro	10	200	True	11110000111100001111000011110000111100001111	10101010101	0
5	opt_s30_t150.log	micro	30	150	True	11110000111100001111000011110000111100001111111	10101010101	0
10	opt_s10_t150_d.log	micro	10	150	False	11110000111100001111000011110000111100001111111	10101010101	0
12	opt_s50_t200.log	micro	50	200	True	11110000111100001111000011110000111100001111	10101010101	0
14	opt_s30_t150_d.log	micro	30	150	False	11110000111100001111000011110000111100001111111	10101010101	0
16	opt_s10_t150.log	micro	10	150	True	11110000111100001111000011110000111100001111111	10101010101	0
18	opt_s10_t200_d.log	micro	10	200	False	11110000111100001111000011110000111100001111	10101010101	0
21	opt_s50_t150_d.log	micro	50	150	False	11110000111100001111000011110000111100001111111	10101010101	0
25	opt_s10_t200.log	one	10	200	True	11110000111100001111000011110000111100001111	10101010101	0
28	opt_s50_t200_d.log	one	50	200	False	11110000111100001111000011110000111100001111	10101010101	0
32	opt_s10_t150_d.log	one	10	150	False	11110000111100001111000011110000111100001111111	10101010101	0
37	opt_s30_t150_d.log	one	30	150	False	11110000111100001111000011110000111100001111111	10101010101	0
39	opt_s10_t150.log	one	10	150	True	11110000111100001111000011110000111100001111111	10101010101	0
41	opt_s10_t200_d.log	one	10	200	False	11110000111100001111000011110000111100001111	10101010101	0
43	opt_s50_t150_d.log	one	50	150	False	11110000111100001111000011110000111100001111111	10101010101	0

05 - Results

Worst 10 results

	file	model	cm	ms	light	message	final_message	LD
44	opt_s30_t50_d.log	one	30	50	False	11110000111100011110001111000111110001111111...	1010101010111111111	7
11	opt_s30_t50.log	micro	30	50	True	11110000111100001111000011110000111100001111111...	1010101010111111111	7
22	opt_s30_t50_d.log	micro	30	50	False	11110000111100001111000011110000111100001111111...	1010101010111111111	7
15	opt_s10_t50_d.log	micro	10	50	False	11110000111100001111000011110000111100001111111...	1010101010111111111	7
38	opt_s10_t50_d.log	one	10	50	False	1110000111110001111000011110000111100001111111...	1010101010111111111	7
26	opt_s50_t50_d.log	one	50	50	False	11110001111100011111000111110001111100011111111...	1010101010111111111	7
4	opt_s50_t50_d.log	micro	50	50	False	11110000111100001111000011110000111100001111111...	1010101010111111111	7
1	opt_s50_t50.log	micro	50	50	True	111111111111111111111111111111	1111111	5
7	opt_s30_t100_d.log	micro	30	100	False	11110000111100001111000011110000111100001111111...	101010101011	2
8	opt_s10_t100.log	micro	10	100	True	11110000111100001111000011110000111100001111111...	101010101011	2

05 - Results


LD absolute frequencies
Frequency
LD
0
2
4
6
8
10
12
14
16
0
1
2
3
4
5
6
7

06

CONCLUSIONS and FUTURE PROSPECTS

Further analyses

06 - Conclusion

- Real time is not so trivial

- Limitations
  - Imposing a start and end message we limit the sending/receiving process
    - If the end message is a sequence of 1 of len N (111...1)
    - We can transmit at maximum N-1 sequences of 1 consecutively
  - High environment sensitivity
    - Cheap material (led, photodiode)

06 - Conclusion

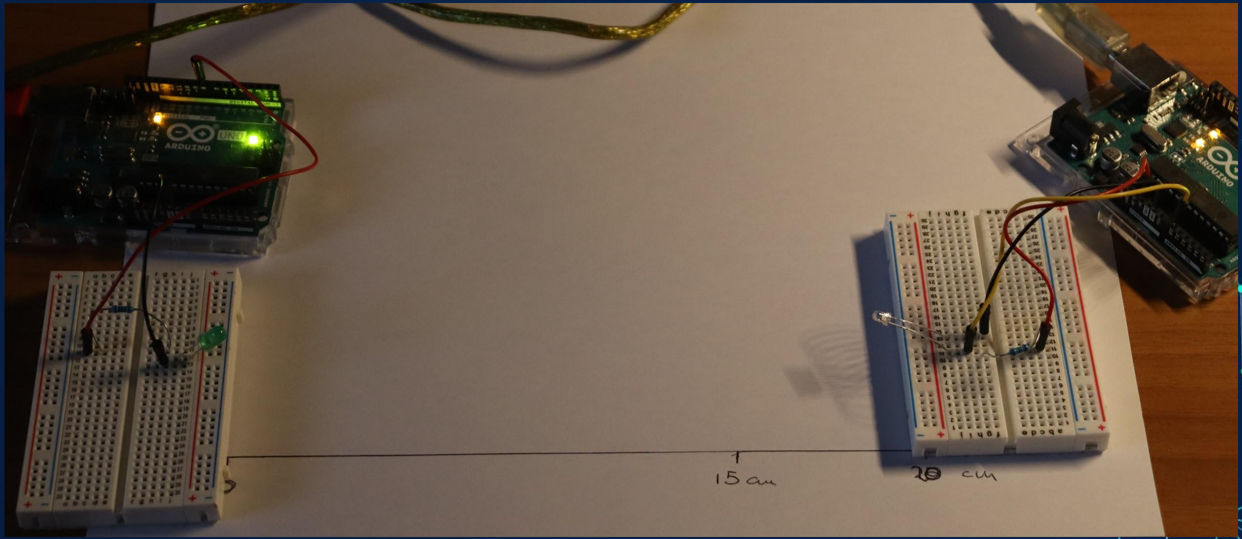
- Next challenges:
    - Improve the algorithm
        - In block code, k=4 is not ideal

            - 1100 ⇒ 1? or 0?
            - Better to use odd numbers (k=3)

    - Export it to a raspberry PI

06 - Do we still have time? :-)
Simulation DEMO

- **Youtube link:** https://youtu.be/sn3vl7yCjGM



THANK YOU