UNIVERSITÉ DE FRIBOURG
UNIVERSITÄT FREIBURG

HUMAN-IST
HUMAN CENTERED INTERACTION SCIENCE & TECHNOLOGY

A simple python cross-platform API for controlling the e-puck2 educational robot

Author : David Roman Frischer[1] Supervisor : Julien Nembrini[2]

September 14, 2021

Department of Informatics - Bachelor Project Report

Département d'Informatique - Departement für Informatik • Université de Fribourg - Universität Freiburg • Boulevard de Pérolles 90 • 1700 Fribourg • Switzerland

phone +41 (26) 300 84 65 fax +41 (26) 300 97 31 Diuf-secr-pe@unifr.ch http://diuf.unifr.ch

[1] david.frischer@unifr.ch, HUMAN-IST group, DIUF, University of Fribourg
[2] julien.nembrini@unifr.ch, HUMAN-IST Department, University of Fribourg

Abstract

The e-puck2 is a small and compact robot designed for education and research purposes. One can control it in many ways: Bluetooth, WiFi, I2C or by simulation with Webots. In the past it has been difficult for students to start working with the API in C language. This report presents an API that gives straight forward methods to control the e-puck2. It is a package written in Python, and available on Pypi.org and comes with documentation on ReadTheDocs.org. Tested by more than thirty students during a semester at the University of Fribourg in 2021, the feedback from the students satisfied the goal of the API. This report also presents the pi-puck, a physical extension for the e-puck2. Its components are described and tested to draw conclusions for its potential use in projects. This report ends with a face detection benchmark to test the performance of the pi-puck. Four different controllers, which differ in the process of transferring and processing the image, are tested. The results of this report contribute to a better understanding of the core of this new API and provide a better understanding of the e-puck2.

Keywords: Bachelor project report, Human-IST Research Institute, e-puck2, pi-puck, face detection, GCtronic, EPFL

Contents

Introduction

What is a robot? According to the definition in the Cambridge dictionary [5], "a robot is a machine controlled by a computer that is used to perform jobs automatically". The use of robots is extensive, including domestic machines, also known as white goods (vacuum cleaners, washing machines, laundry machines, etc.), to the industrial world (sorting, transportation, security, etc.), to the health sector (surgery machines, assistive robotics, and machines, etc.), and of course in the economic sector (self-checkout machines, ATMs[1], trading bots, etc.). Thanks to progress in the use of robotics, industrial productivity and efficiency have increased radically during these last decades because robots can work continuously without getting tired.

Robots are becoming smarter every day with better machine learning algorithms and faster training process. As reported recently, Boston Dynamics has started to sell their robots in many sectors such as military, space, security and health[1]. However, those who do not understand the technology behind what may appear to be just pieces of metal and gadgets, are still afraid to adopt it. For example, on April 29, 2021, the New York Police Department had to return their purchase from Boston Dynamic because they were receiving complaints from citizens concerning anxiety provoked from the use of a robotic dog[6].

Despite some ongoing reservations, the world of robotics will continue to progress and be a source of fascination to the outside world. The next big steps in the consumer world will likely be in the automotive industry, with improvements in self-driving cars, and the adoption of more human-like domestic robots and other forms of Artificial Intelligence in our homes.

Anyone who would like to start in this blossoming field needs to understand the challenges and the limits of the robotics domain. Knowing how to write simple lines of code and having patience are essential prerequisites to getting started in this field. Having easy to use tools to facilitate learning is also extremely important. Accessible robots for beginners, such as Thymio by Mobsya or Mindstorms by LEGO, are available on the market. The Thymio can be controlled by aligning statement logic and actions blocks. It is a fine approach for young kids whereas a good approach for young teenagers would be Mindstorms is excellent. The Mindstorms robot has to be built with LEGO blocks that can be made into five different combinations. Whereas the Thymio is only controlled by aligning code, the Mindstorm additionally gives the possibility to use any programming language to be able to write more advanced algorithms.

A more advanced beginner's robot, and one which is particularly good for junior developers to start with, is the e-puck[2]. The e-puck was developed by the École Polytechnique Fédérale de Lausanne (EPFL) in collaboration with GCtronic. Thanks to its small size and its simplicity of use, it allows a smooth approach to discover the world of robotics. The e-puck can only be controlled with programming languages or with the ROS[3] software. The University of Fribourg has used the e-puck first generation robots for its first-year course in the Computer Science Bachelor syllabus, for a number of years. In 2018, the e-puck2, an improvement to the original e-puck, was introduced with new possibilities to enhance the user experience[4].

The e-puck2 can be controlled using different platforms. The two platforms used at the University of Fribourg are simulation in a computer and, control by WiFi. Both control platforms used the same application programming interface (API) made by Dr. Julien Nembrini and written in C language[5]. Dr Nembrini's API provided the methods for both platforms, but the manner to call these methods had the potential to be improved. The aim to offer students a more efficient structure encouraged us to write a new cross-platform API where the user can control the robots in multiple ways without modifying the controller of the robot. This new API is written in Python which is easy to learn, provides scientific analytic tools used in the course and offers a variety of open-source packages. This report will first discuss the e-puck2, and the improvements it has over its predecessor. Next, we will describe our tailor-made API which was developed to control the e-puck2 via WiFi, I2C bus and simulation. Then, we will discuss and analyse the pi-puck, a physical extension of the e-puck2, and outline its pros and cons. Finally, we will compare the performances

[1] Self-checkout and ATM's comply by definition
[2] http://www.e-puck.org
[3] www.ros.org/
[4] More information in section 1
[5] Github link : github.com/nembrinj/epuckAPI

of the e-puck2 controlled via WiFi and controlled via the pi-puck. The comparisons will be done with the help of a face detection algorithm. The analyses will measure their speed to transfer and process information and images. With these results, we will be able to draw conclusions on the potential use of the pi-puck in future projects.

1 About the e-puck2

GCtronic released the e-puck2 in 2018. Compared to its first version, improvements were made without significantly changing its mechatronics design. Both versions have two wheel motors, eight infrared proximity sensors, three ground sensors, a CMOS camera, LEDs around the PCB and, many more features that are shown in Figure 1 and 2.

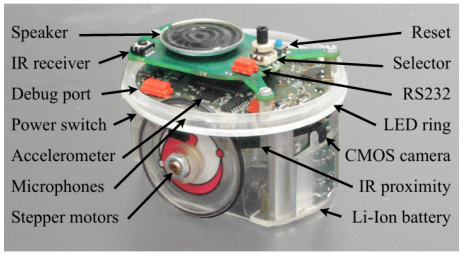

Figure 1: e-puck1 [4]

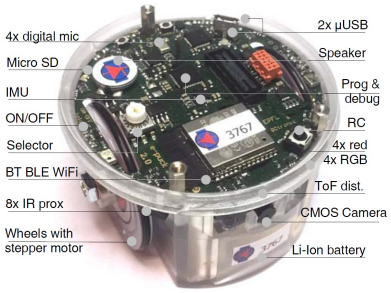

Figure 2: e-puck2 [3]

Figure 1 and 2 respectively show the first and second version of the robot. Some visual differences are noticeable. On the superior part of the e-puck1, its green light PCB with a speaker and a selector has been removed from the e-puck2 and been replaced with new components directly on its main motherboard.

1.1 New on the e-puck2

The e-puck2 offers new features such as WiFi communication with the computer, a better camera resolution thanks to its better CPU, a Time of Flight sensor and, a micro-USB port to charge the battery directly on the robot instead of pulling it out to charge it on a dock. The shape of the e-puck2 is more ergonomic than its predecessor to allow multiple extensions to be stacked. Table 1 illustrates the biggest improvements of the e-puck2 compared to its first generation.

Feature	e-puck1	e-puck2
Battery	Only chargeable with an external charger	Chargeable directly on the e-puck with a micro-USB type B
Processor	16-bit dsPIC30F6014A @ 60MHz (15 MIPS)	32-bit STM32F407 @ 168 MHz (210 DMIPS)
Memory	RAM : 8 KB; Flash : 144 KB	RAM : 192 KB; Flash : 1024 KB
Distance sensor	8 infra-red sensors measuring proximity up to 6 centimetres	same infra-red sensors, ToF sensor up to 2 metres
Camera	VGA colour camera; 52x39 pixels	same camera; 160x120 pixels
LEDs	8 red LEDs around the robots	4 red LEDs and 4 RGB LEDs
Communication	RS232 and Bluetooth 2.0 for connection and programming	micro-USB Type B, Bluetooth 2.0, WiFi 2.4GHz

Table 1: Differences between e-puck version 1 and 2
Source: https://www.gctronic.com

E-puck first and second generation do not show significant differences, but still gives us the opportunity to do better image processing at a 160x120 pixels resolution. Also, thanks to the micro-USB charger, it reduces the risk of breaking the contact point between the battery and the robot. Regarding the price, the e-puck2 remains the same as the e-puck1.

1.2 Advantages with WiFi

The WiFi on the e-puck2 is a good improvement on the robot. Today, WiFi is one of the most standard protocols for communication between devices. Below, are the advantages that WiFi brings to the e-puck2.

- Instantaneous connection with the robots. (Bluetooth requires a pairing process)
- TCP protocol for reliable communication of data
- Faster exchange of packets (approx. 20Mbit/s [2])
- Larger range of communication (Limited by the the WLAN access point)

1.3 Sending and receiving WiFi packets

An e-puck2 connected via WiFi to a Personal Computer (PC) uses the TCP protocol, where the robot is listening and the PC makes requests to it. Once the connection is established, the robot waits for the computer to send a packet of bytes. On reception, it responds with another packet of bytes. The TCP protocol automatically takes care to transmit reliable data with no errors. Figure 3 is a simple schema to illustrate how the e-puck2 is connected the PC via WiFi[6].

Figure 3: E-puck2 connected with the intracommunication of the API via WiFi and controlled by a PC

1.3.1 Sending a WiFi packet

	Request	Settings	Motors				LEDs	RGB LEDs				Speaker
	Flags	Flags	Left LSB	Left MSB	Right LSB	Right MSB	State	LED1	LED3	LED5	LED7	Sound id
(# bytes)	(1)	(1)	(1)	(1)	(1)	(1)	(1)	(3)	(3)	(3)	(3)	(1)
address:	1	2	3	4	5	6	7	8-10	11-13	14-16	17-19	20

Table 2: WiFi packet sent from the computer to the robot
Source: Gctronic.com

Table 2, describes the WiFi packet sent to the e-puck2 from a PC. The last line of the table is the address of each element of the packet. For example, the addresses from 8 to 10 of the packet sets the RGB LED1 of the robot. Each address corresponds to a byte to set the RGB value where each byte corresponds to a color of the RGB model. From the start of the packet, the first address of the packet is to enable and disable the camera and sensors by the shift of a bit. The second address is for the sensor calibration and the position and speed of the motors. The remaining addresses are there to set the motors and LEDs, and finally the last address is to select a melody to be played on the speaker.

[6]It is possible to find a detailed description of the communication protocol development on the GCtronic web page www.gctronic.com

1.3.2 Receiving a WiFi packet of data sensors

The e-puck2 can send back two different types of packets. One packet is for feedback from the sensors (with a value of `0x02` on the header of the packet) and the other one is for image feedback (with a value of `0x01` on the header). By knowing the header value, the parse of the received packet can be done correctly. Table 3 illustrates a packet of the sensors's feedback received by the computer from the robot.

	IMU							IR sensors		ToF	
(# bytes)	Acc (1)	Acceleration (4)	Orientation (4)	Inclination (4)	Gyro (6)	Magnetometer (12)	Temp (1)	8 x proximity (16)	8x ambient (16)	Distance (2)	...
address:	0-5	6-9	10-13	14-17	18-23	24-35	36	37-52	53-68	69-70	

Microphones	Motors		Battery	uSD	TV remote			Selector	Ground sensor		Button	Reserved
4 x volumes (8)	Left steps (2)	Right steps (2)	ADC value (2)	State (1)	Toggle (1)	Address (1)	Data (1)	Position (1)	3 x proximity (6)	3 x ambient (6)	State (1)	Empty (1)
71-78	79-80	81-82	83-84	85	86	87	88	89	90-96	97-102	103	104

Table 3: WiFi packet sent from the robot to the computer
Source: Gctronic.com

To convert the data from bytes to a number, we use the `struct` [7] module. The module allows us do byte conversions in a single line of code for each part of the packet.

```
# 2 bytes per sensor, odd position is LSB and even position is MSB
PROX_SENSORS_COUNT = 8
prox_values = [struct.unpack( "H", struct.pack("<BB",
                wifi_packet[37+2*i],
                wifi_packet[38+2*i]))[0]
                for i in range(PROX_SENSORS_COUNT)]
```

Listing 1: Converting the proximity values from two raw bytes to an integer

The snippet code in listing 1 demonstrates how the `struct` package converts the proximity sensor values from bytes to integers. The composition of the variable `wifi_packet` is the same as in table 3. Each proximity sensor uses two bytes and the addresses of the eight sensors are from 37 to 52.

1.3.3 Receiving a packet of an image

As mentioned in section 1.3.2, the header value for the packet containing an image is `0x01`. The packet size is 38400 bytes[8] in a RGB565 format. Due to the big size of the image, the e-puck2 can only send it in multiple packets. Once all the packets have been received, a conversion from RGB565 to BGR888 is necessary to once again convert the image to a bitmap format for the user.

[7] https://docs.python.org/3/library/struct.html
[8] The 38400 bytes image is the multiplication of $160 * 120 * 2$. Where 160 is the width of the image, 120 is the height of the image and we use 2 channels.

1.4 Webots configurations

Webots simulation offers the possibility to control an e-puck2. To use the robot in Webots, a new e-puck Node found in the `robots/gctronic` folder must first be added. Next, the configuration of the Node, must be changed from version 1 to version 2 and set the camera and set the camera to a better resolution. Figure 4 shows the window to set the characteristics of the robot in Webots.

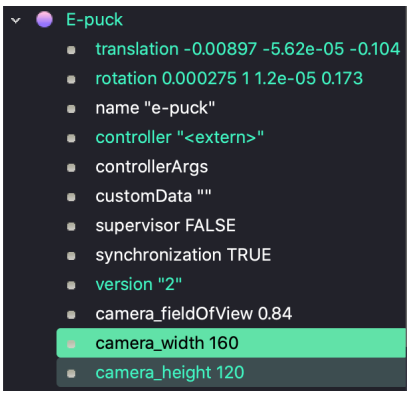

Figure 4: Camera size on Webots

1.4.1 Running Python with Webots simulation

In order to run the Python controller with Webots, the `PYTHONOPATH` must be set. It needs to know where to find the Python folder to be able to run Python files.

```
$ which python3
```

Listing 2: Find Python path

The command in listing 2 can be run from the Terminal to find the currently used Python3 folder. Once found, the Python path should be pasted into the Webots configuration, as shown in figure 5

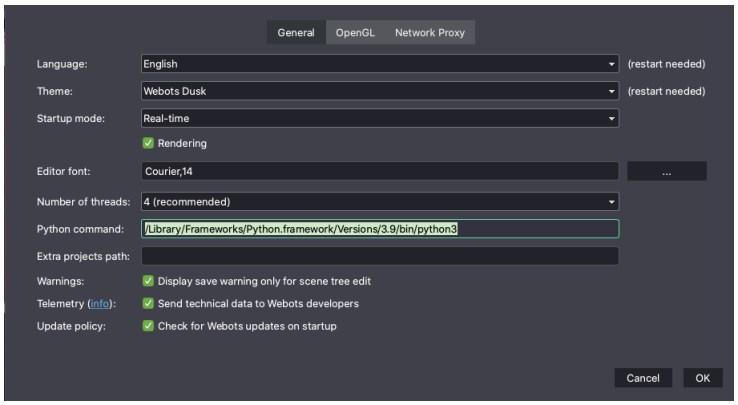

Figure 5: Configure the Python path in Webots

1.4.2 Running a Simulation from the Terminal

It is also possible to start a Webots simulation from the Terminal. The following instructions (1. to 4.) allow for the necessary configurations to run a simulation from the terminal:

1. Change the controller
   Set the controller to extern in the e-puck node. The figure 6 illustrates the modification.

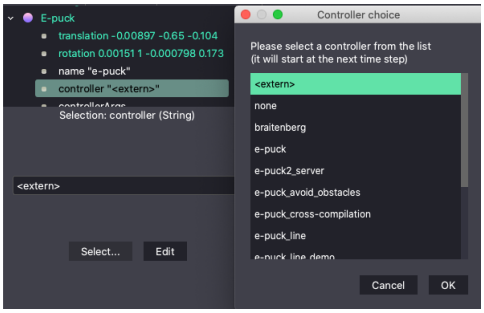

Figure 6: Select the extern controller

2. Modify the PYTHONPATH
   The PYTHONPATH must be configured in such a way that Python can start the simulation on Webots.

   - On MacOS 10.15
     Edit the .zshrc file and add the Webots PYTHONPATH

```
nano ~/.zshrc
```

     Insert the following lines in the file. The PYTHONPATH should correspond with your current Python3.x version.

```
export WEBOTS_HOME="/Applications/Webots.app"
#Using Python3.9
export PYTHONPATH="${WEBOTS_HOME}/lib/controller/python39"
```

   - On Ubuntu
     Edit the .bashrc file and add the Webots PYTHONPATH

```
nano ~/.bashrc
```

     Insert the following lines in the file. The PYTHONPATH should correspond with your current Python3.x version.

```
export WEBOTS_HOME="/usr/local/webots"
#Using Python3.8
export PYTHONPATH="${WEBOTS_HOME}/lib/controller/python38"
```

3. Restart the Terminal and the world simulation, and finally run the Python script for each e-puck2 on your Terminal.

1.5 Discussion

While working with the e-puck2, it was noticed that the ESP-32 sometimes has trouble working fluently due to a high demand of tasks (TCP communication, capturing sensor and image data, receiving and sending packets). Overwhelming the e-puck2 with demands, over a very short time, can lead to a short freeze of the CPU. To avoid this issue, one can disable the camera of the e-puck2 when it is not being used and enable it again as required.

A further matter that was noticed, after establishing exactly ten new connections via WiFi in a row, was that the e-puck2 could no longer establish further WiFi connections with the computer. This obstacle could only be rectified by manually rebooting the robot. The cause for this break in communication is unknown and could either be a security function from the firmware or simply a malfunction.

One more issue that was noticed, concerned the Time of Flight sensor. Unfortunately, how the sensor has been embedded on the robot, it needs to be checked. The feedback from this sensor is particularly sensitive and, depending on the axe of the sensor, can provide varying results. Figure 7, shows the ToF sensor oriented 20 degrees higher than the horizontal axe.

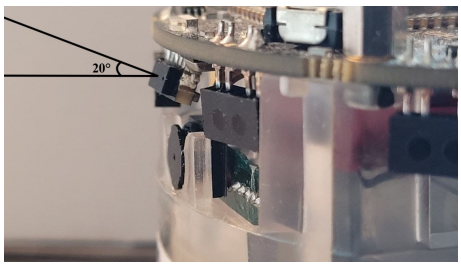

Figure 7: Axe of the ToF sensor

The magnetometer and the infra-red ambient light sensor do not seem to work on the e-puck2, which was already a defect noticed on the first e-puck version.

Finally thanks to a better CPU, the camera resolution of the robot was significantly increased. The advantage of having a more performant camera resolution potentially opens up possibilities for future projects with a camera, which may require a more efficient object follower or perhaps object or face recognition and detection.

2 The cross-platform API

In this section, the structure and the contents of the API will be described. Then chosen snippet codes in C and Python will be compared to show the advantages of the new API and finally the challenges encountered during the project will be discussed.

2.1 Motivation

Developing in C can be challenging, especially for beginners. Being immediately introduced to pointers, file structure, structures, memory allocation and compilation, may overwhelm students with nonessential subjects for the robotics course. Our motivation was to avoid unnecessary complications and so that students could focus on writing controllers for the robots. Using Python allows the student to ignore a multitude of C subjects and gives the following advantages:

- Simplifies and reduces the learning curve for the use of robots.
- Thanks to the programming-oriented object paradigm, the same code can be used to control the robots, regardless of the environment (WiFi, simulation, RaspberryPi).
- Many free packages are available for heavy computing and analysis (numpy, pandas, matplotlib, ...)

The unifr_api_epuck package has been written for students in a way that they can start controlling the robots in less than five minutes.

2.2 File structure

The package contains multiple folders and files. In the root of it, one can find the wrapper.py file and three folders. The wrapper file provides instances of the e-puck2. The first folder contains the scripts to control the e-puck2 for the three platforms (WiFi, Webots and pi-puck[9]), the second folder contains the scripts for the graphical user interface and the third folder for the intercommunication program between the robots. Figure 8 illustrates the structure of the package.

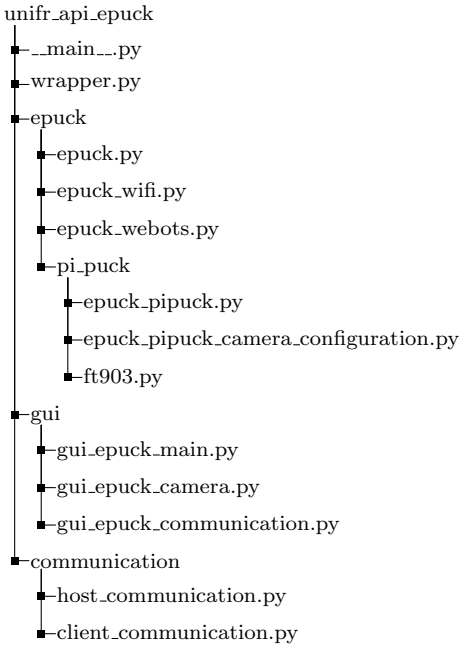

Figure 8: Tree Structure of the Package

[9]More information about the pi-puck in section 4

2.2.1 wrapper.py

The wrapper is the starting point to get an instance of an e-puck2. Depending on the arguments in the method `wrapper.get_robot(args)`, it will decide in which environment the robot is used[10]. The wrapper can give an instance of an e-puck2 controlled by WiFi, by a pi-puck or a simulated e-puck2 in Webots. Unfortunately, it is not possible for the wrapper to get an instance of an e-puck2 that is controlled via Bluetooth. It was decided to not implement it because of its unreliable connections and to its slow data transfer service with a PC.

The wrapper also gives the possibility to any device that can run Python3 to be connected to the intercommunication service provided by the API[11]. In order to make this connection, one needs to put in arguments, an arbitrary chosen unique id and then the IP address of the host. Listing 3 provides an example.

```
1 from unifr_api_epuck import wrapper
  #insert an unique id name
3 my_computer = wrapper.get_client('my_computer_105')
  #connect to the IP address of the communication host
5 my_computer.init_client_communication('192.168.112.24')
```

Listing 3: Instance of a client PC to communicate with robots

The snippet code in listing 3 shows the variable `my_computer` which represents an instance of a PC. At line 5, `my_computer` connects to the communication host to be able to communicate with the robots using the intercommunication service of the API.

2.2.2 E-puck sub-module

The e-puck module contains all the Python files to control the robot. In the following lines, each module will be described.

- epuck.py
  The e-puck.py contains the `EPUCK` class. The latter can be defined as an abstract class that the `WifiEpuck`, `WebotsEpuck` and `PipuckEpuck` classes will inherit. The intercommunication is already implemented in the `EPUCK` class.

- epuck_wifi.py
  The Python file epuck_wifi.py contains the class `WifiEpuck` which inherits of the `EPUCK` class. It has all the algorithms for sending and receiving the WiFi packets[12] via the TCP protocol between the robot and the computer.

- epuck_webots.py
  The Python file epuck_webots.py contains the class `WebotsEpuck`. Similar to `WifiEpuck` class, it inherits of the `EPUCK` class. The `WebotsEpuck` communicates differently with the computer. It uses the provided methods from the Webots API[13].

- sub-module pi_puck
  The sub-module pi_puck contains multiple Python files. It is modelled in a more complex structure than the epuck_webots.py and the epuck_wifi.py file because the Raspberry Pi and the pi-puck interact with buses which require configurations when starting the robot.

  - epuck_pipuck.py
    The epuck_pipuck.py contains the `PipuckEpuck` class. This class inherits from the `EPUCK` class and has the same methods as the `WifiEpuck` and `WebotsEpuck`. The `PipuckEpuck`

[10] More information on the decision tree to get an e-puck2 instance in section 2.4
[11] More information on the intercommunication service in section 2.6.2
[12] More information on the intracommunication protocol in section 1.3
[13] Webots API : cyberbotics.com/doc

class uses the `smbus`[14] Python package to interact between the robot and the embedded Raspberry Pi on the pi-puck.

- epuck_pipuck_camera_configuration.py
  The epuck_pipuck_camera_configuration.py is necessary when getting the control of the camera's robot. This file will enable the possibility to interact with the camera's e-puck such as it was an embedded webcam of the Raspberry-Pi. This will lead us to use OpenCV to get the images.
- ft903.py
  The ft903.py is a necessary Python file to control the pi-puck LEDs linked to the ft903L chip.

2.2.3 GUI sub-module

The GUI sub-module contains all the Python files for the Graphic User Interface. It is implemented with the Tkinter packet for a fast, portable, and reliable experience for the user. The gui_epuck_main.py file contains the code for the main window of the GUI. From the main window, two services are possible. One service is to visualise the images taken from the robot and another is to create a host for intercommunication between devices. The gui_epuck_communication.py and gui_epuck_camera.py contain the code to run these two services[15].

2.2.4 communication module

The communication module contains two files. One is the program for the communication host and the second one is for the client to be able to connect to the host.

- The host_communication.py
  The host_communication.py contains the `EpuckCommunicationManager` class which inherits of `SyncManager` from the multiprocessing[16] packet. The `EpuckCommunicationManager` will create and host a communication between all of the robots (and PCs) independently of their environment.
- client_communication.py
  The client_communication.py contains all the necessary methods for the communication service. When implementing a new object that will use the communication, this object should create an instance of the `ClientCommunication` from the client_communication.py. For example, the class EPUCK uses an instance of `ClientCommunication`.

2.3 Differences between C language and Python

The following comparisons will show that writing Python controllers will be more concised, easier and innate for the developer.

2.3.1 Getting the control of an e-puck2

The simple example in listing 4 shows how to start controlling a robot in C and in Python. With C, the conditional compilation macro is used to include the necessary header file. It will decide which robot it needs between a simulated or a real to control the e-puck2. Then, the robot can be controlled with the given methods of the API in C. In Python, the `wrapper` from the `unifr_api_epuck` package returns an instance of the robot by considering the arguments in the `get_robot(args)` method.

[14] More information about the smbus2 package : pypi.org/project/smbus2
[15] More information on the GUI in section 2.5 and 2.6
[16] https://docs.python.org/3/library/multiprocessing.html

C

```c
#define SIMULATION 0

#if SIMULATION
#include "../API/webots/webotsAPI.h"
#else
#include "../API/epuck/epuckAPI.h"
#endif

void robot_setup() {
    init_robot();
}

int main (int argc, char **argv) {
    #if SIMULATION
    #else
    ip = argv[1];
    #endif

    robot_setup();
}
```

Python

```python
from unifr_api_epuck import wrapper
import sys

if __name__ == "__main__":
    ip_addr = None
    if len(sys.argv) == 2:
        ip_addr = sys.argv[1]

    my_robot = wrapper.get_robot(
        ip_addr)
```

Listing 4: Comparisons in C and Python to get control of an e-puck2

2.3.2 Using the Object Oriented Paradigm

One of the most significant changes in the new coding style is the use of the object oriented paradigm such that the robot instance can be controlled by calling its methods.

As an example, the codes in C and Python in listing 5 show how to update the sensor values with the `go_on()` method and get the proximity sensor values on the next instruction. The Python controller retrieves the updated ground sensors values by calling `my_robot.get_ground()` rather than assigning the value in the variable `ground_values` by using pointers in C.

C

```c
short int ground_values[
    GROUND_SENSORS_COUNT];
robot_go_on();
get_ground(ground_values);
```

Python

```python
my_robot.go_on()
ground_values = my_robot.get_ground()
```

Listing 5: Retrieving the sensor values with the API in C or Python

2.3.3 A Simple Example

The example in listing 6 shows the codes for the robot to go forward and backward repeatedly every 50 loops in C and Python. It completes the codes in listing 4 to get a fully operational example controller for the e-puck2.

C

```c
#define SIMULATION 0
#if SIMULATION
#include "../API/webots/webotsAPI.h"
#else
#include "../API/epuck/epuckAPI.h"
#endif
void robot_setup() {
    init_robot();
    init_sensors();
}
void robot_loop() {
    int counter = 0;
    while (robot_go_on()) {
      int dir = (++counter % 100) >
          50 ? -1 : 1;
      set_speed(dir * NORM_SPEED, dir
          * NORM_SPEED);
    }
    cleanup_robot();
}
int main (int argc, char **argv) {
    #if SIMULATION
    #else
    ip = argv[1];
    #endif

    robot_setup();
    robot_loop();
}
```

Python

```python
from unifr_api_epuck import wrapper

def main(ip_addr):
    robot = wrapper.get_robot(ip_addr
        )
    NORM_SPEED = 2
    counter = 0

    while robot.go_on():
        counter += 1
        dir = 1 if counter \% 50 >
            100 else -1
        robot.set_speed(dir*
            NORM_SPEED)

    robot.clean_up()

if __name__ == "__main__":
    ip_addr = None
    if len(sys.argv) == 2:
        ip_addr = sys.argv[1]

    main(ip_addr)
```

Listing 6: Controller example for the e-puck2 with the API in C or Python

The C code uses predefined methods that are linked to the robot being controlled. Python code calls the methods of the robot instance. As an example in line 11 of the Python code, the instance calls the `set_speed(left_speed_motor, right_speed_motor=None)` method to modify the wheels speed of the robot.

2.4 Getting the correct instance of the e-puck

The API uses the factory method pattern to create instances of the robot. The factory pattern is in the wrapper.py file in the unifr_api_epuck API and deduces which instance to create by inserting arguments put in `get_robot(ip_addr, is_pipuck)` method.

Figure 9 illustrates what the factory goes through to decide which instance of the e-puck2 to return. The first decision is if the robot is controlled by a pi-puck. The default value is `False` which means that the user has to write `is_pipuck = True` to tell the factory that the e-puck2 will be controlled by a pi-puck. The second decision is to see if there is or not, an IP address of the robot. If not, logically the WiFi is not necessary which brings to the conclusion that the end user will be using Webots to simulate the e-puck2 otherwise, it will return an e-puck2 instance controlled by WiFi.

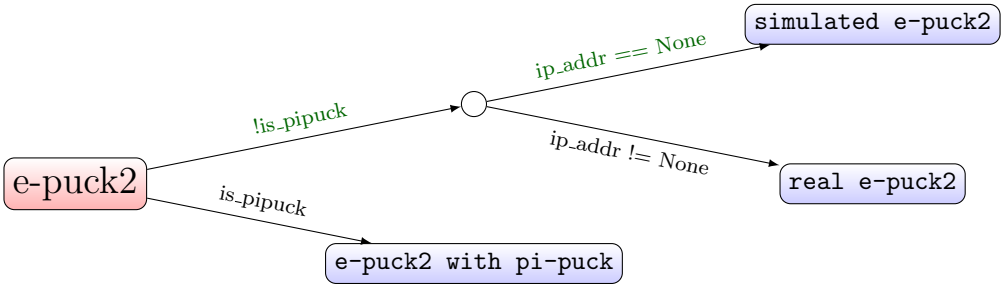

Figure 9: Decision Tree at instantiate

2.5 The Graphic User Interface

During the development of the API, it was important to check whether messages were correctly processed. It was also important to continuously monitor the message pending for each robot. As Tkinter was already used for the live stream, it was decided to combine a monitor communication and a live stream tool to make a graphic user interface. Figure 10 shows the main window of the GUI. From this point, one can either start the host communication (top green rectangle) or start a live stream (bottom green rectangle).

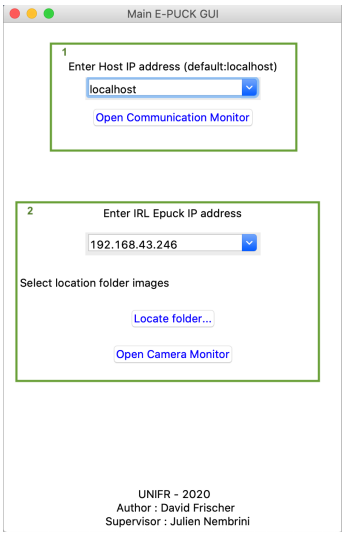

Figure 10: Main window of the graphic user interface

When a host communication is started, the monitor of the host communication window opens as shown in figure 11. The communication monitor shares information such as, whether the host is online, which clients are online and how many pending messages they have. Additionally, messages can be sent to all the robots or to a specific e-puck2. In order to do this, the user has to select the recipient from the drop-down list, write the message and press "enter" to send it. When an e-puck2 disconnects from the host manager, the latter will remove it from the list of the GUI. Unfortunately, it is not possible to monitor the communication on multiple devices at the same time. The reason for that is to not overload the access of the database which would unnecessarily slow down the users.

Figure 12 shows the window of the live stream. A slider can be used here to adjust the refresh rate and the image will automatically adjust itself depending on the window size. A picture from the GUI can also be taken. Finally, the window gives information from which robot it is being streamed and indicates the path of the folder where the pictures have been saved[17].

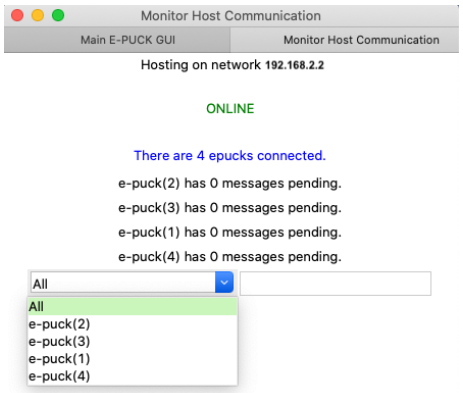

Figure 11: Host Communication GUI

Figure 12: Live stream GUI

2.6 Challenges during development

The Python package was predominately inspired by the C API package developed by Dr. Julien Nembrini[18]made in 2019. During the development of the Python package, many challenges occurred. Smaller problems, which will be discussed later, included:

- Find the right process to decode the packets for the intracommunication between the e-puck2 and the computer[19].
- Fast conversions and manipulations of the images that came from the robot.
- Learning to create a documentation on readTheDocs.org and upload the package on Pypi

The above obstacles were not hard to correct but much time was needed to gather the information and learn what needed to be done to make it work. More of a stumbling block were two greater difficulties which were firstly, the ability to visualise, from the PC, a live image from the robot's camera and secondly, writing a stable intercommunication between the devices.

[17]More information about the live stream in section 2.6.1
[18]https://github.com/nembrinj/epuckAPI
[19]More information in section: 1.3

2.6.1 Live Streaming

The goal of live stream is to visualise on the spot the point of view of the robot. It also enables the developer to be assured that the camera works correctly and in a fast manner. Live streaming is not possible with the C API.

The live stream was first implemented using Pygame[20] package. Pygame is well known for creating video games on Python. Even though using Pygame was successful, several issues made it clumsy to work with. Impediments included the heavy package of the Pygame, the high amount of CPU needed to use it, and the boot time of the controller being too slow due to the initialization that Pygame needed to do before running the code.

Next, an attempt to enable live streaming was made using Tkinter[21]. Tkinter is well known for writing simple and light graphic interface independently of the operating system. The first version attempted using Tkinter seemed convincing in every aspect in comparison to the approach using Pygame.

A problem that arose during the implementation with Tkinter, that was already present with Pygame, was that the stream had to be run in a separate main process. As a result, it was not possible to create a GUI from an e-puck2 controller.

Saving the images from the robot was another problem that needed to be solved. This was achieved by saving each image in a specific folder and the Graphic User Interface made with Tkinter had to retrieve it from the same place to display it on its frame. The stream was finally made possible by saving the new image using the same name as the old one (in order to overwrite it) and then by refreshing the frames that followed, thus turning it into a series of moving images.

Figure 13 illustrates the chronology of an image transfer from the e-puck2. Once the e-puck2 captures the image, the controller running on the computer will retrieve the image from the robot and save it in a local folder. In the case where the image already exists in the folder, it will replace it. Then the GUI will go find this image in the same folder to display it on its screen. With this process, the GUI just has to refresh the image multiple times in a second to create a video from it. The key of this stream, it's the fact that the new image has the same file name. This will ease the process to change the picture rapidly.

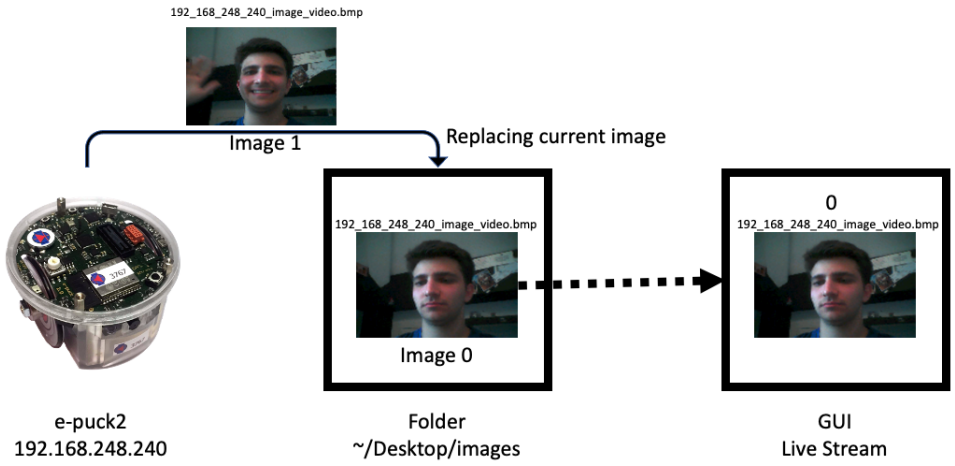

Figure 13: File Transfer for the live stream

[20] https://pypi.org/project/pygame/
[21] https://docs.python.org/fr/3/library/tkinter.html

2.6.2 Communication between the robots

Before, communication between the e-puck robots in the old API used a shared memory in the computer that controlled them. This meant that all the e-pucks had to be controlled from a single computer. Memory space was limited, and due to C language restrictions, users had to transform the message into a packet of bytes to be sent and then unpacked, to be read. Another disadvantage with C, was that the memory space had to be constantly allocated and freed for a reliable communication.

Using Python, solutions were found to solve these problems. SyncManager[22] is used to host a "mailbox" for the robots. Each computer that controls an e-puck will connect to SyncManager via a socket connection. SyncManager keeps a dictionary to save the messages received for the connected robots. The snippet code in listing 7, is an example of the "mailbox".

```
{'connected':{'192.168.40.191':True,
              '192.168.40.122':True},
'192.168.40.191' : ['Hi', 123, [1.96,3.62,5.53]],
'192.168.40.122' : []
}
```

Listing 7: A Python dictionary to manage the intercommunication from SyncManager

The first key of the dictionary: `connected`, keeps a sub-dictionary of all the robots online. During the communication, SyncManager puts all the robots flags to `False` and within the following five seconds, the robots have to put back their flag to `True`. They update their flag in the dictionary when the method `my_robot.go_on()` is called. If after the time limit is passed and its value is still `False`, SyncManager considers that the robot is offline and deletes the messages destined for it. The second and third key, in the figure, are the robots IP addresses. The values of the keys are their pending messages. A robot retrieves its messages in a FIFO algorithm.

Figure 14 illustrates the two e-pucks of listing 7. They are connected via WiFi to their respective computers; Computers One and Three. As these computers control the robots, they also have to manage the intercommunication for them. As illustrated in figure 14, the computers are on the same local network of Computer Two. Computer Two plays the role of the host manager[23].

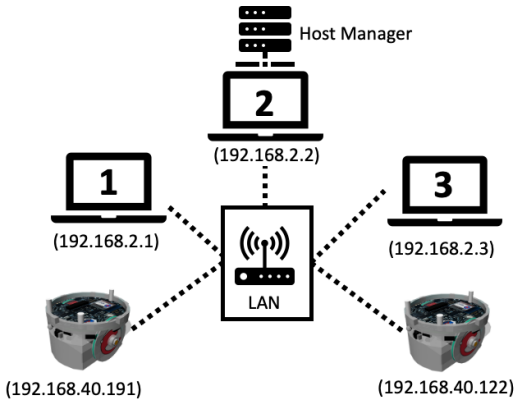

Figure 14: Star topology intercommunication between the robots

Using a central SyncManager, as was done with Computer Two, suggests that a "Star Topology" for the intercommunication is being used since all the computers which communicate between themselves, have to do so by connecting to a central computer. As seen in section 2.6.1, this topology does not apply for the live stream because the images stay in the local memory.

[22] https://docs.python.org/3/library/multiprocessing.html
[23] Any computer can be the host manager. For this scenario, Computer Two is the chosen host.

There are many advantages to this new communication system. One of the advantage is that if a robot disconnects, it will not break the communication for the other robots because all the data is kept on the host computer. On the other hand, if a problem arises with the host computer and this independent process gets disturbed, the messages of all the robots risk being lost. Another advantage is that any type of data can be sent such as a string, an array or even an image. As mentioned earlier, sending other types of data was not possible before, with the old API system. A further advantage is that the memory storage of the host manager is dynamically taken care of by Python and the developer does not have to deal with it anymore. The user can also send messages to any other recipient that can run Python program.

3 Deployment of the API

Once the API was ready, its deployment and maintenance had to be easy. Additionally, students should be able to access to the package documentation online.

3.1 The source code

3.1.1 Sharing the code

It was decided to use the Pypi.org servers to host the package. Pypi.org is already well known to host, for free, some of the most famous packages available such as Numpy, Pendelum, PyQT, Pandas and many more.

To upload the API on Pypi, the online course made by realpython.org : "How to Publish Your Own Python Package to PyPI" made by Joe Tatusko was a good tutorial to follow. In less than two hours, the package was online and ready to be downloaded in any operating system with a simple `pip3 install unifr_api_epuck` command.

3.1.2 Maintaining the code

This section gives instructions for developers who will maintain the source code of the unifr_api_epuck package. Eligible access is required to use the manager and update the code on Pypi.org.

First time:

1. The user must install wheel[24] and twine[25] packages

```
$ pip3 install wheel twine
```

2. Git pull the source code of the unifr_api_epuck package from Github[26]

Every time the user updates the package:

4. The user must modify the setup.py file at the root of the folder and increase the version number.

5. Finally, from their Terminal at the root of the repository folder, the user must update the package on Pypi with the help of the makefile command:

```
$ make update
```

The makefile command will automatically delete the old build and unifr_api_epuck.egg-info version to replace it with new ones. Then the user will have to enter their Pypi credentials to be able to finalise the upload. Finally, the user must not forget to push the changes on GitHub to keep consistency for other developers.

3.2 The documentation

3.2.1 Sharing the documentation

It was decided to use the server of ReadTheDocs.org to host the official documentation of the package. ReadTheDocs is as well-known as Pypi.org, and is used to share documentation online. The official documentation is available on unifr-api-epuck.readthedocs.io. The documentation contains: a first steps tutorial to start the API within a few minutes, ideal for beginners with the API. It also has a fully detailed documentation for the controls of the robot, useful to go get to know all the robot's possible methods. More advanced examples for the camera and the communication are available in the examples section.

[24] https://pypi.org/project/wheel
[25] https://pypi.org/project/twine/
[26] https://github.com/davidfrisch/UNIFR_API_EPUCK

3.2.2 Maintaining the documentation

This section is for developers who will maintain the documentation of the unifr_api_epuck package. It necessitates an eligible access of the GitHub repository.

The documentation of the source code is written in docstring directly in the source code. The other files are written in a reStructuredText format. They can be found at this following path : `UNIFR_API_EPUCK/docs/`

- first_steps.rst is the first steps tutorial to start with the API.
- graphic_user_interface.rst is an explanation for the GUI.
- `unifr_api_epuck` folder contains mostly files that are used to convert the docString to a reStructuredText text using the auto-generated tool with Sphinx[27] .
- `res` folder contains all the images used in the documentation
- `example` folder contains examples of how to use the camera and the intercommunication.

Once the documentation is revised, the user has just to commit and push the changes to the GitHub repository. ReadTheDocs will automatically clone it to its server to apply the modifications.

[27] sphinx-doc/autodoc.html

4 The Pi-Puck Extension

The pi-puck is an extension for the e-puck designed by York Robot Lab in collaboration with GCtronic. The extension allows a Raspberry Pi to be mounted on the e-puck2[28].

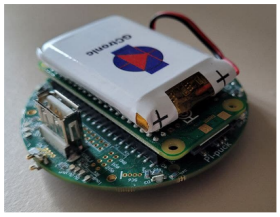

Figure 15: Front side

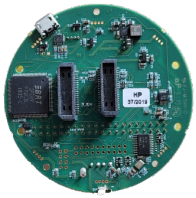

Figure 16: Back side

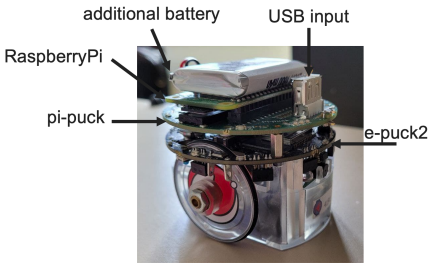

Figure 17: Pi-puck on e-puck2

The pi-puck has three RGB LEDs and a FT903L microcontroller to control them, a speaker, a microphone, an expansion header to attach the Raspberry Pi, and an additional battery.

4.1 First Steps with the pi-puck

The pi-puck can easily be stacked on the e-puck2. The Raspberry Pi is normally already stacked on the pi-puck at delivery.

4.1.1 Installing the York firmware on the e-puck2

To use the pi-puck with the e-puck2, a customized firmware on the e-puck2 is necessary. The firmware is downloadable on the pi-puck webpage on gctronic.com. One must follow the instructions on the e-puck2 webpage on gctronic.com to update the firmware on the robot.

After the installation process, we noticed that unfortunately the WiFi control for the robot does not work anymore on selector position 10 and 15. The reason for this problem is still unknown. The only way to make it work again, is to install the original firmware back on the e-puck2.

4.1.2 Configuring the WiFi

The Raspberry Pi can be connected to the WiFi. One way to configure it, is to connect it to a monitor with the help of the mini HDMI output. Once connected, the `wpa_supplicant.conf` file can be edited from the Terminal or text editor with the help of the monitor and a keyboard.

The user can edit the file from the Terminal with the following command:

```
$ sudo nano etc/wpa_supplicant/wpa_supplicant.conf
```

The user must modify the following lines according to their WiFi settings.

```
ctrl_interface=DIR=/var/run/wpa_supplicant GROUP=netdev
update_config=1
country=CH

network={
        ssid="MySuperWiFi"
        psk="Batman123"
}
```

Finally, Raspberry Pi must be restarted to connect to WiFi.

[28] Pi-puck website: https://www.gctronic.com

4.1.3 Modifying files in the Raspberry Pi with scp

The Raspberry Pi can have a SSH access to be controlled remotely. The user will need to enable the SSH mode on the Raspberry Pi.

To enable the SSH on the Raspberry Pi, the user must open the Raspbery Pi configuration:

```
$ sudo raspi-config
```

To enable the SSH the user must in the raspi-config select: `3 Interfacing Options / P2 SSH` and finally choose `"Yes"`. It is possible to control remotely using VNC[29] which is already installed by York University on the Raspberry Pi.

4.1.4 Transferring files without a password

By default, each time one uses the scp command, the user must input a password to grant the data transfer. Using a public key and private key is an easy solution to avoid inserting a password at each scp command request. To do this, it is recommended to follow the tutorial: "How to use the Linux 'scp' command without a password to make remote backups" from Alvin Alexander. Our personal configuration was that the Raspberry Pi generated the keys and our PC had the public key.

4.2 Starting the controller at the boot time of the Raspberry Pi

To be able to start the robot's controller immediately after turning on the pi-puck and the Raspberry Pi, the user can install a `.service` file[30] from the UNIFR_API_EPUCK GitHub.

The user can execute the following command in the Raspberry Pi Terminal to install the repository at its root.

```
$ cd ~
$ git clone https://github.com/davidfrisch/UNIFR_API_EPUCK.git
```

To install the `.service` file, the user simply executes the following command at the root of the unifr_api_epuck folder.

```
$ make service
```

It will install as the sudo user the `pipuck-start.service` in the `/etc/systemd/system` folder of the pi-puck.

Once correctly installed, the service will execute only one time the `my_controller.py` file from the `/UNIFR_API_EPUCK/pi-puck-conf` folder at boot time.

To modify the controller, one must modify the `my_controller.py` file and run the `make service` script again.

To stop the service, the user must execute the following command

```
$ sudo systemctl disable pipuck-start.service
```

To enable it again, the user must write the following command :

```
$ sudo systemctl enable pipuck-start.service
```

[29] Download VNC on your computer from www.realvnc.com
[30] www.raspberrypi.org

4.3 First example with the pi-puck

To activate the pi-puck, the user must first enter the Python example code given in section 2.3.1. Then the Boolean `True` must be added in argument in the `wrapper.get_robot(...)` to let the wrapper understand that the e-puck is using a pi-puck. Listing 8 shows the snippet code to get an instance of an e-puck2 with the pi-puck.

```
from unifr_api_epuck import wrapper
import sys

if __name__ == "__main__":
    ip_addr = None
    if len(sys.argv) == 2:
        ip_addr = sys.argv[1]

    my_robot = wrapper.get_robot(ip_addr, True)
```

Listing 8: Creates the instance of an e-puck2 in Python with the pi-puck

The IP address is not necessary for the intracommunication between the e-puck2 and the pi-puck. On the other hand, it is necessary if we need to enable the intercommunication between the robots.

Figure 18 illustrates the connections between the devices. The Raspberry Pi (the red rectangle on the pi-puck) controls the e-puck2 via the pi-puck, directly attached between them. The PC is only used to send the robot's controller file or to remotely control the Raspberry Pi with a remote service (SSH, VNC Viewer, ...).

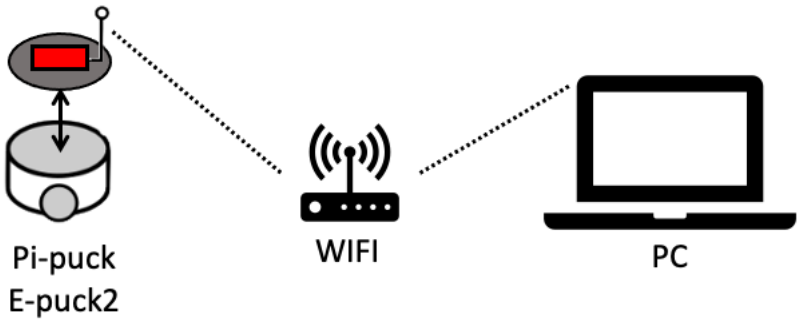

Figure 18: E-puck controlled by a pi-puck and, Raspberry Pi remotely controlled by a PC.

4.4 The embedded speaker and microphone on the pi-puck

The embedded speaker and microphone are connected to the Raspberry Pi. It is possible to play sound with the command `aplay` for .wav files or `mplayer` for .mp3 files from the Terminal. One can also record sound from the pi-puck using the `arecord` command.

```
arecord -Dmic_mono -c1 -r16000 -fS32_LE -twav -d8 test.wav
```

After multiple trials, we failed to emit a loud sound from the pi-puck but succeeded to play in good quality with an external computer. Additionally, an unpleasant ultrasound came out of the pi-puck once the microphone or loudspeaker was active.

The microphone records a good quality sound when using the `arecord` command. An attempt was also made to record sound with Python using the playsound package. Unfortunately, errors occurred due to ALSA device detection problems.

4.5 The Time of Flight sensor on the pi-puck

The Raspberry Pi requires the VL53L0X package in its python environment to use the Time of Flight sensor.

It can be installed with the following command:

```
$ pip3 install git+https://github.com/gctronic/VL53L0X_rasp_python
```

Finally, to test, a simple Python script can be written for the pi-puck. For example:

```
from unifr_api_epuck_test import wrapper
r = wrapper.get_robot('192.168.105.98', is_pipuck=True)
r.init_tof()

while r.go_on():
    tof_sensor = r.get_tof()
    print(tof_sensor)
```

4.6 The e-puck2 camera with the pi-puck

The e-puck2 camera acts like a webcam on the Raspberry Pi. The `my_robot.go_on()` method does not need to be called to capture an image. The unifr_api_epuck uses the OpenCV2 package to capture and manipulate pictures taken from the robot. These images are saved in a .jpg format and not in a .bmp format like in the other platforms.

During the experiments with the camera, a delay was seen due to a buffer. This implied that the Raspberry Pi was keeping an action in its memory that had already passed a few seconds ago rather capturing a live picture. Following this observation it was needed to find a way to empty the buffer to get the most recent live action. The solution found was to take a small series of pictures (five pictures in the unifr_api_epuck API) just before sending it to the Raspberry Pi.

4.6.1 Special required files for the camera

To activate the camera with the Raspberry Pi, the unifr_api_epuck API automatically takes care of the camera configuration. At the the initialisation of the epuck's camera, the `pipuck_camera_configuration.py` is run. This execution can be seen on listing 9 from line 5 to 7.

```
from .epuck_pipuck_camera_configuration import main as
    main_cam_configuration
#(...)
def init_camera(self, folder_save_img = None, size= (None,None)):
    #(...)
    cam_init_thread = Thread(target=main_cam_configuration, args=())
    cam_init_thread.start()
    cam_init_thread.join()
    self.camera = cv2.VideoCapture(0)
```

Listing 9: Enabling the e-puck2's camera for the pi-puck

During the execution of the `pipuck_camera_configuration.py` file, the executable `camera-configuration` Unix file as shown in listing 10 on line 9, processes. This file will detect which type of camera the e-puck2 has to configure with the pi-puck.

These two files detect and configure the necessary parameters to enable the camera. Normally, the Raspberry Pi that comes with the pi-puck already has the executable Unix file in `/home/pi/Pi-puck/`. If it is missing, it can be copied from the unifr_api_epuck repository in the pi-puck-conf folder. The pi-puck-conf folder must be copied in the Raspberry Pi memory in `/home/pi/Pi-puck/`[31].

[31] If necessary, create the missing folders.

```
#this is the epuck_pipuck_camera_configuration file
#(...)
def main():
    poXXXX_detected = False
    ov7670_detected = False
    bus = smbus2.SMBus(I2C_CHANNEL)
    try:
#detect the embedded camera of the e-puck2 with the Unix executable file
        p = subprocess.call("/home/pi/Pi-puck/camera-configuration")
#(...)
```

Listing 10: Detecting the e-puck2's camera model with the necessary files for the pi-puck.

4.7 Differences in performance between WiFi and pi-puck

In this section, we will discuss the performance differences between the pi-puck and the WiFi on the e-puck2. The three noticeable differences are the initialisation time of the controller for the robot, the capture time of an image and finally, the exchange speed between the robot and its controller.

4.7.1 Initialisation time

The initialisation time is the moment from the start of the script until the robot has completed its initialisation. The robot can either be controlled via WiFi or with the pi-puck via I2C bus. Even though the initialisation time only happens at the start of the process, a significant difference of time between the platforms may be noticed.

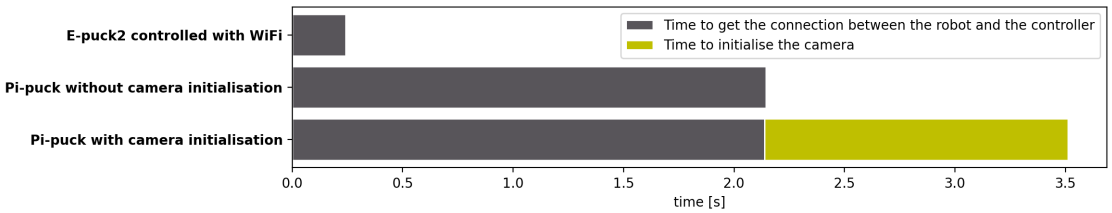

Figure 19: Time to start the robot's controller

Figure 19 shows the average of ten samples for each platform with the robot. The robot takes 0.242 seconds with WiFi to establish its TCP connection to the PC. The pi-puck takes at least 2.144 seconds and with the camera it takes 1.373 seconds more to start the robot. It takes more time to initialise the pi-puck with the camera because of the two scripts, mentioned in section 4.6, that are needed to configure the camera to be a webcam on the Raspberry Pi[32]. Figures 20 to 22 illustrates figure 19 with box plots.

[32]For the time being, this is unavoidable as both scripts will always be executed at the start of the program when the API is used to retrieve the instance of an e-puck2 controlled by a pi-puck.

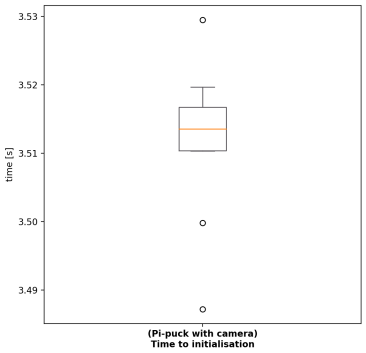

Figure 20: Pi-puck – Time to start the robot's controller with the camera

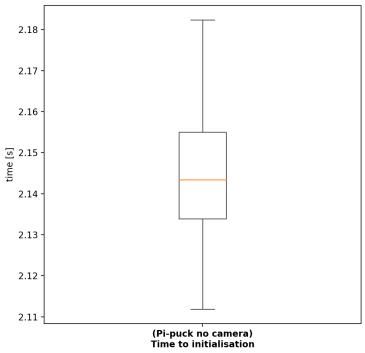

Figure 21: Pi-puck – Time to start the robot's controller without the camera

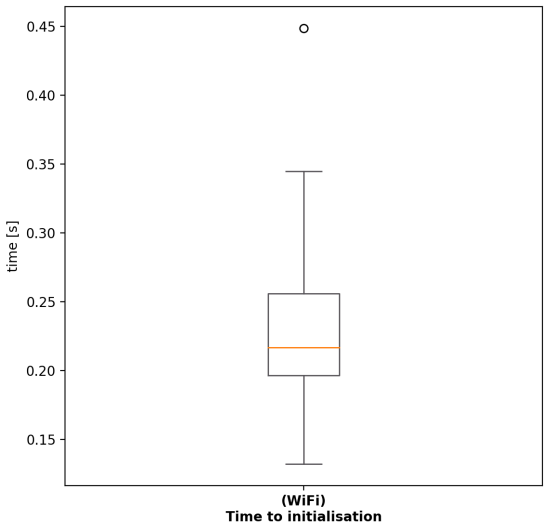

Figure 22: WiFi – Time to start the robot's controller

4.7.2 Time to exchange data between computer and robot

One exchange of data is defined as the cycle to send and receive data a single time from the controller to the e-puck2. An exchange of data happens when the user calls the method `my_robot.go_on()`. Even though the name of the method is the same for the pi-puck and the WiFi control, they still remain different in their respective protocol. The left list below shows the steps with a pi-puck controlled via the I2C bus and the right list below shows the different steps of the exchange from a PC to an e-puck2 controlled via WiFi.

Steps via I2C bus

1. Create a checksum
2. Read and Write the data sensors
3. Check the checksum from the reading of the robot
4. A time sleep for stability reason for the pi-puck (not mandatory)

Steps via WiFi

1. Send data from PC to robot
2. Receive the packet from the robot
   (a) Check the header of the packet
   (b) Retrieve the image from the image packet
   (c) Retrieve the sensor data from the image packet

The sleep time in step 4 via I2C bus is not mandatory. The unifr_api_epuck API uses it because the official GCtronic example also uses it. Listing 11 is a snippet code from the GCtronic official GitHub.

```
#Communication frequency at 20 Hz.
time_diff = time.time() - start
if time_diff < 0.050:
    time.sleep(0.050 - time_diff)
```

Listing 11: Time sleep from an official example from line 167 to 170
Source: github.com/gctronic

After running performance tests, figure 23 shows the results of performance tests ran via WiFi to control the e-puck2 and figure 24 gives the cycle speed in detail for the e-puck2 controlled with the pi-puck. One can see that, surprisingly, it takes a considerable amount of time to check the data header and with less surprise, the image packet is the one that takes the longest to process. A data via WiFi is nearly eight times faster when the camera is disabled.

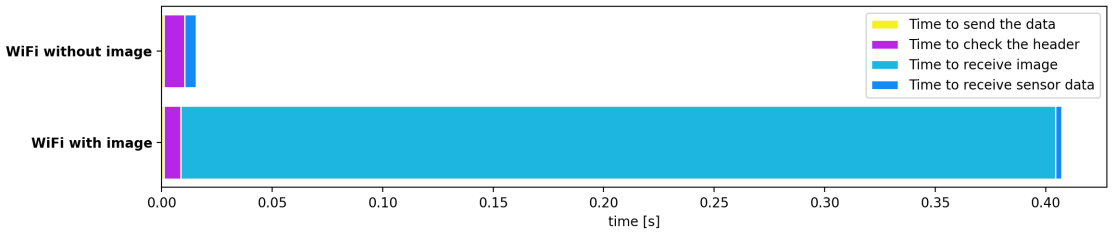

Figure 23: Frequency exchange data of an e-puck2 by WiFi

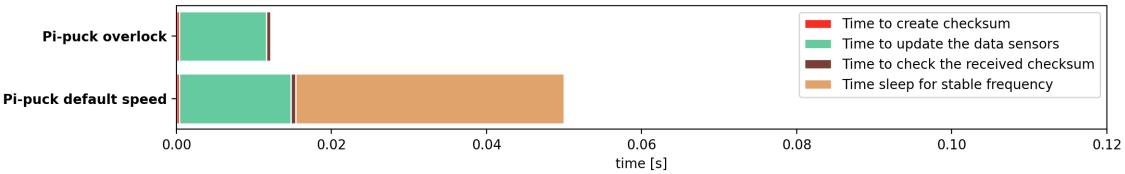

Figure 24: Frequency exchange data of an e-puck2 with the pi-puck

To summarise figures 24, and 23, the fastest cycle can be reached at approximately 85Hz with the pi-puck[33], which is four times and a quarter faster than the default clock speed with the pi-puck at 20Hz. The speed when the pi-puck is overclock is so fast for the exchange that it sometimes lead to read errors. Fortunately, there is a support in the unifr_api_epuck API to handle this situation to avoid interrupting the program. The second fastest cycle is with WiFi, with the camera disabled, the exchange speed reaches at approximately 68Hz. When the camera is enabled with WiFi, the frequency of the cycle is at approximately 2.46Hz. This is 27.6 slower than the 68Hz exchange using WiFi with the camera disabled. For more in-depth details, figure 25 to 28 are the box plots of figure 23 (WiFi) and figure 29 to 31 are the box plots of figure 24 (Pi-puck).

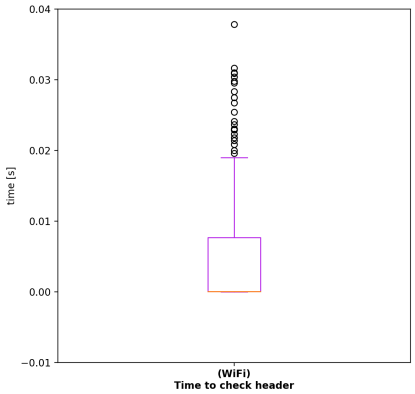

Figure 25: WiFi – Time to check header

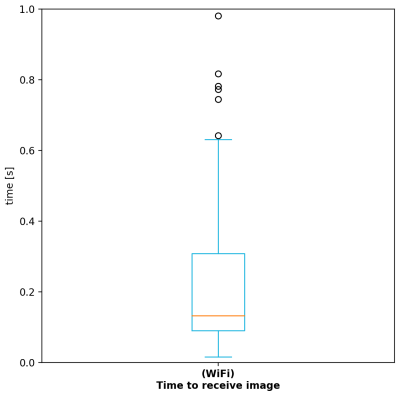

Figure 26: WiFi – Time to receive image

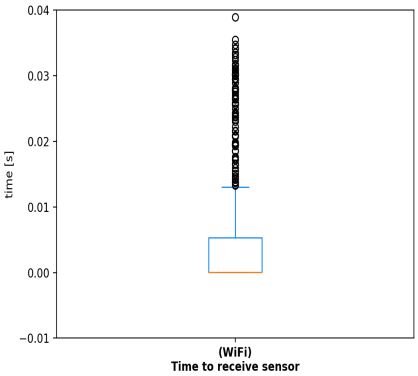

Figure 27: WiFi – Time to receive sensors

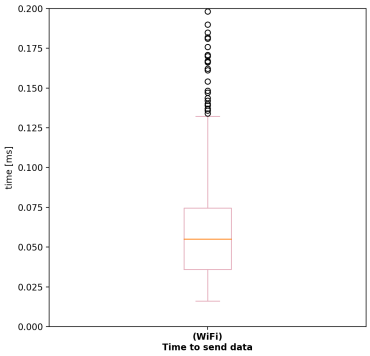

Figure 28: WiFi – Time to send data

[33]Note that the cycle on the pi-puck does not consider the capture of an image because the method my_robot.go_on() does not retrieve the image during the exchange.

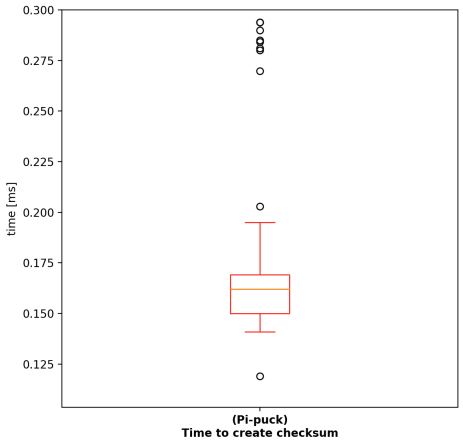

Figure 29: Pi-puck – Time to create checksum

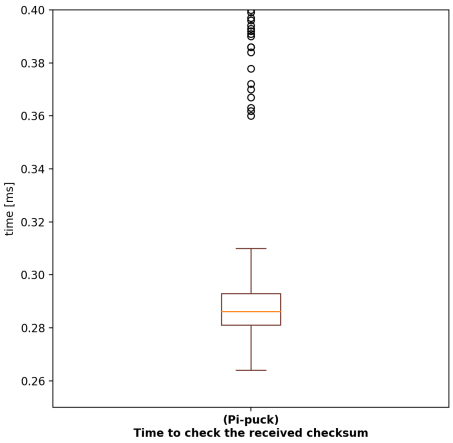

Figure 30: Pi-puck – Time to check checksum

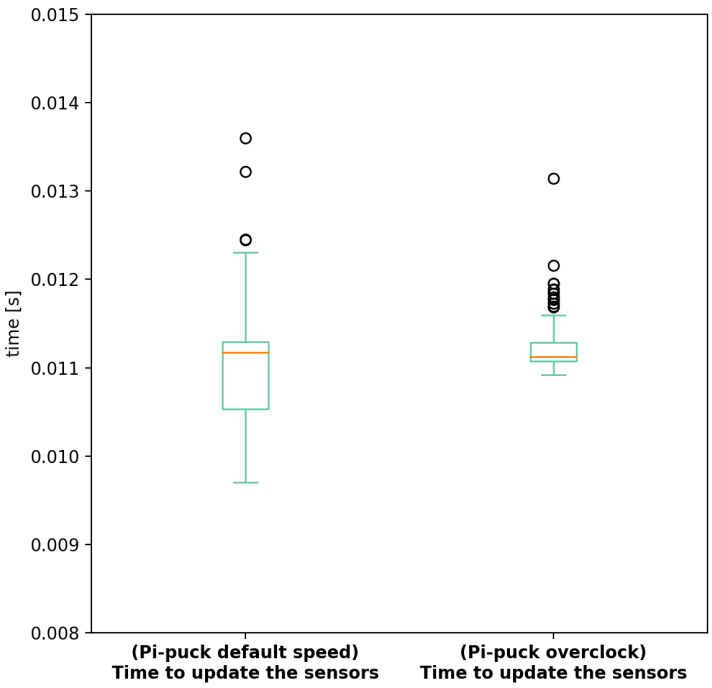

Figure 31: Pi-puck – Time to update sensors

4.7.3 Time to take a picture

The process of capturing an image from the e-puck2 consists of the following three steps:

1. Capture and receive the image from the e-puck2
2. Process the image (convert the format or resize it)
3. Write the image in the PC disk (not mandatory)

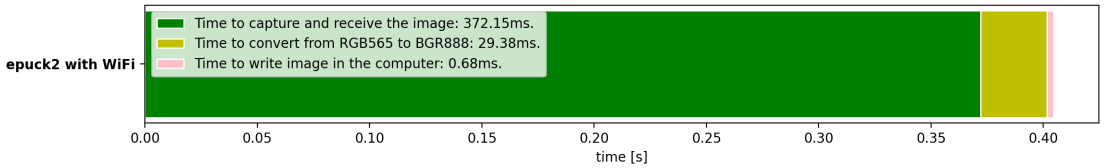

Figure 32: Time to capture an image using the e-puck2 with WiFi

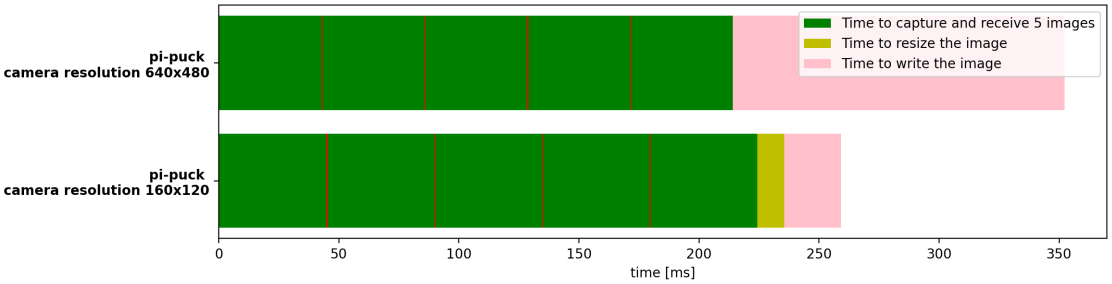

Figure 33: Time to capture an image using the e-puck2 with the pi-puck

The time required to take a picture with the e-puck2 from the pi-puck, and from the e-puck2 controlled via WiFi, respectively was measured and results are illustrated on figure 32 and 33.

To capture and receive one image is quicker with the pi-puck, with an average compared to WiFi. Even if the pi-puck is faster than the WiFi, it can be improved. As explained in section 4.6, the API needs to take several pictures in a row to update its buffer to get the most current action.

The time for image processing is different depending on the platform. The e-puck2 controlled via WiFi takes 29.38 ms to convert its RGB565 image to a BGR888 image. If the e-puck2 is controlled via the pi-puck, it is possible to resize the image for better performance in the overall process. As shown in figure 33, it is faster to reduce the size of the image and then write it to the Raspberry Pi disk, than to leave the original image and save it directly. Overall, taking a picture with the e-puck2 with saving it to disk, using the pi-puck can be done in about 351.35 ms without resizing the image, 259.04ms with resize and, via WiFi, it will take approximately 402.21 ms. For more in-depth details, figure 34 to 36 are the box plots of figure 32 (WiFi) and from figure 37 to 41 are the box plots of figure 33 (Pi-puck).

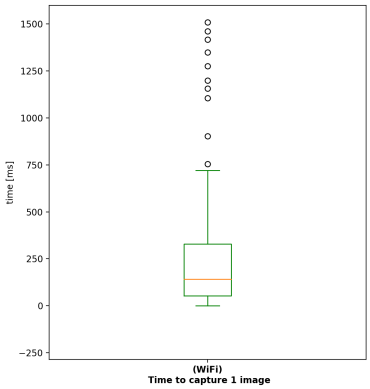

Figure 34: WiFi – Time to take image

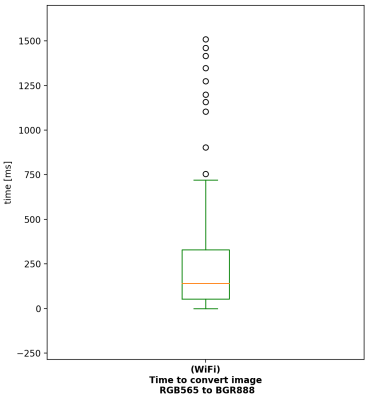

Figure 35: WiFi – Time to convert image

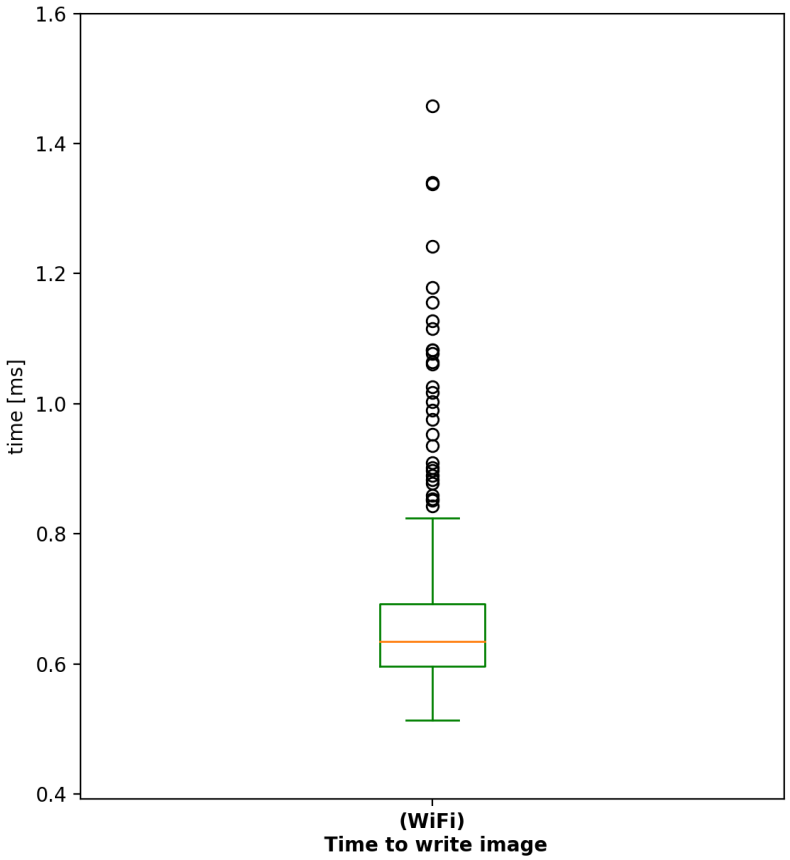

Figure 36: WiFi – Time to write image

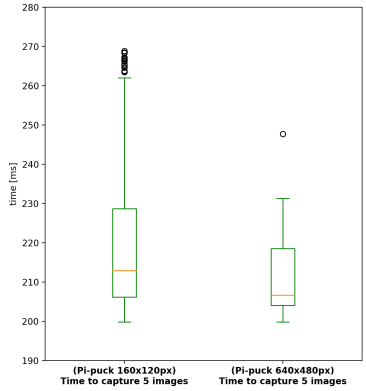

Figure 37: Pi-puck – Time to take image

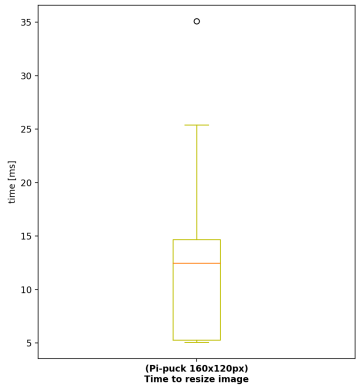

Figure 38: Pi-puck – Time to resize image

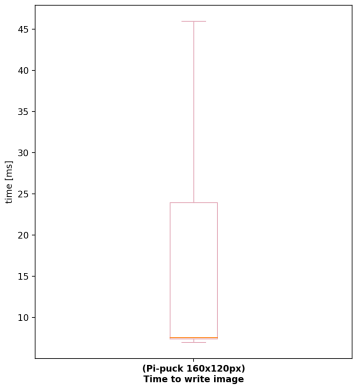

Figure 39: Pi-puck–Time to write small image

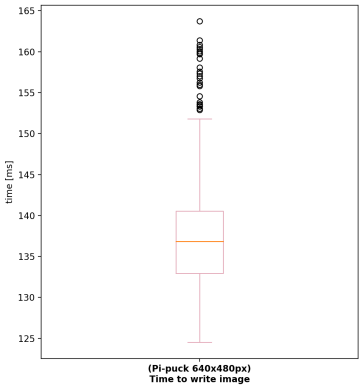

Figure 40: Pi-puck–Time to write big image

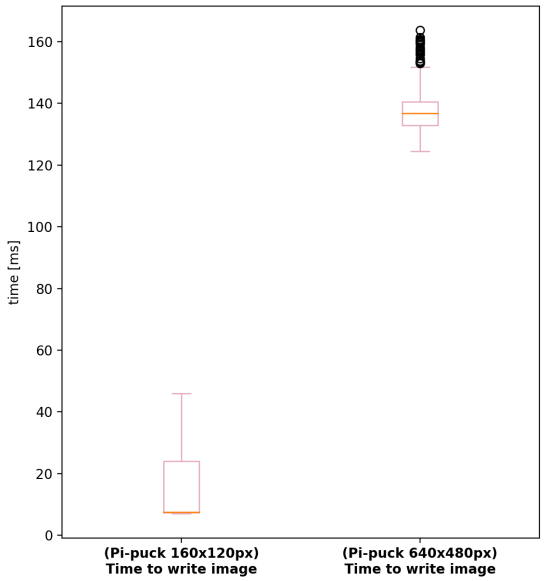

Figure 41: Pi-puck – Comparison time to write image between the two resolutions

5 Testing the pi-puck with a face detection algorithm

The face detection algorithm was used to test performance. Giving the robot the ability to detect faces was used to make a comparison between using the pi-puck and using WiFi to control an e-puck2. The Viola-Jones classifier was used to enable face detection and from it, be able to calculate the distance between the centre of the camera and the face, to adjust the robot's position.

5.1 Usage of The Viola-Jones classifier

The Viola-Jones classifier[34], is a famous object detection framework published in 2001. OpenCV offers multiple pre-trained classifiers trained on faces to be able to detect them. For this report, the haarcascade_frontalface_default.xml classifier and the OpenCV2 methods are used to detect the face as shown in listing 12.

```
# take the image from the e-puck2
original_image = img
#transform the color image in black and white
grayscage_image = cv.cvtColor(original_image, cv.COLOR_BGR2GRAY)
#take the Viola-Jones classifier
face_cascade = cv.CascadeClassifier("../haarcascade_frontalface_default.xml")
#detect the faces with the classifier, returns arrays of the positions of the faces.
detected_faces = face_cascade.detectMultiScale(grayscage_image, 1.3)
```

Listing 12: Code to detect faces from an image using OpenCV and the Viola-Jones classifier

Thanks to its light and simple algorithm, it should be possible to detect many faces in a short time for a PC.

5.2 Moving the robot depending on the face position.

The next algorithm instructs the robot to turn around itself until it is aligned with the detected face. Once the face is detected, it computes the distance between the centre of the image and the centre of the face. If the distance is too far, the robot will turn slightly to correct the difference. Listing 13 shows the instructions for the robot to turn to the right if the value is negative or to the left if the value is positive and will not move if the difference is lower than the tolerance, defined as the `CENTER_TOLERANCE`.

The algorithm of listing 13 can be improved in several ways. For example, one could better adjust the tolerance of the difference between the centre of the robot and the centre of the face. In the example, the camera size is 160x120. However, if one uses a larger image, the tolerance of the centre should be higher. Another aspect that can be improved is the rotation time of the robot. Instead of being hard-coded to rotate for one second, it should depend on the distance between the two objects.

[34] wikipedia.org/Viola-Jones-object-detection-framework

```
CENTRE_TOLERANCE = 10
#check if there is at least one detected face
if len(detected_faces)>0:
    print('face detected')
    #distance difference between the centre of the robot's webcam and the
        first face detected
    diff_x, _ = get_diff(detected_faces[0], grayscage_image)

    #move the robot right or left depending on the position of the face
    if diff_x < -CENTRE_TOLERANCE:
        r.set_speed(0.5,-0.5)
    elif diff_x > CENTRE_TOLERANCE:
        r.set_speed(-0.5,0.5)
    else:
        r.set_speed(0)

#robot moves for 1 second according to the feedback
r.sleep(1)
r.set_speed(0)
#stop the movement of the robot
r.go_on()
```

Listing 13: Instructions to the robot according to the distance between the centre of the image and the centre of the face

5.3 The scenarios variants

Four different versions of the scenario were written to test the performance of the robot using the code in listing 12 and listing 13. Each version has its own way of sending and processing the image. The different scenarios are described in the following paragraphs.

5.3.1 Only with pi-puck

This first scenario uses only the robot, the pi-puck and the Raspberry Pi. No external computer is used during the execution. The robot takes the picture, then the Raspberry Pi uses the Viola-Jones classifier to detect the face and finally the robot moves according to the feedback received from the Raspberry Pi. The benchmark is performed with two different camera configurations. The first one is with an image size of 640x480 and the other one with an image size of 160x120.

5.3.2 E-puck2 controlled from a PC via WiFi

This scenario uses a PC that controls an e-puck2 via WiFi. It is the same code as the scenario in subsection 5.3.1, the only difference is the arguments of `wrapper.get_robot(ip_addr, is_pipuck=True)` method. The e-puck2 via WiFi only works with an image size of 160x120 px.

5.3.3 Pi-puck – intercommunication – PC

This third scenario uses two different Python scripts. The first script is for the Raspberry Pi and the second is for the PC. The sequence is as follows: The RaspberryPi, which controls the e-puck2, takes a picture of the face. Then it sends it to the PC via the intercommunication system created by the unifr_api_epuck packet. Finally, once the image is received by the PC, the PC detects the face and tells the RaspberryPi in return which direction the robot should go.

5.3.4 Pi-puck – scp command – PC

This fourth scenario works in the same way as the scenario in section 5.3.3, the difference being the transmission of the image from the Raspberry Pi to the PC. Instead of the image being sent

through the intercommunication system, it is sent through the scp command[35]. The scp command is preconfigured so that no password is required to authorise the transmission. Once the photo is saved in the PC, the Raspberry Pi signals the PC to run the face detection algorithm. Finally, the PC tells the Raspberry Pi which direction the e-puck2 should turn.

Scenarios variants	Controls the e-puck2	Transmission of the image	Detection of the face	Communication betweeen PC and Raspberry Pi
Only with pi-puck	pi-puck	None	Raspberry Pi	No
E-puck2 controlled from a PC via WiFi	PC	API intracommunication via WiFi	PC	No
Pi-puck – intercommunication – PC	pi-puck	API intercommunication system	PC	Yes
Pi-puck – scp command – PC	pi-puck	scp command	PC	Yes

Table 4: Summary of the scenarios

5.4 Description of the run

Each scenario lasts 60 seconds. Two performances are scored: The first is the time it takes for the robot to take the photo and save it in the storage of the device that controls the robot. The second is the time taken for the photo to enter the device that performs the face detection algorithm until the robot receives feedback.

5.5 Description of the materials

The following devices are used for the benchmarks:

- E-puck2, fully charged, with the number 4191 from the University of Fribourg.
- MacBookPro Late 2013 (PC), running on macOS Catalina with a 2,6 GHz Intel Core i5.
- Pi-puck purchased in 2021.
- Raspberry Pi Zero W : 1GHz, single-core CPU, 512MB RAM and 802.11 b/g/n wireless LAN
- WiFi router is a Samsung Galaxy Fan Edition 5G phone on Android 11.

5.6 Setting up the scenarios

All scenarios run as follows: an e-puck2 is placed nine centimetres away from a 7x5 centimetre image of a face. Then the program runs for sixty seconds. During execution, the e-puck2 rotates to align its camera with the face. Once aligned, the robot remains stationary until the end of the program but continues to process the face detection.

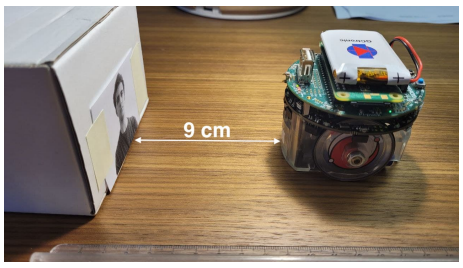

Figure 42: e-puck2 at 9 cm from the face

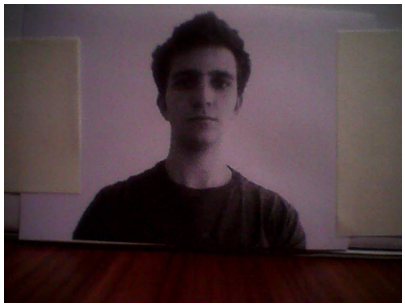

Figure 43: Point of View of the e-puck2

[35]SCP (secure copy) is a command-line utility that allows you to securely copy files and directories between two locations.(linuxize.com)

Figure 42 shows the position of the e-puck2 at the end of a scenario. Figure 43 shows a capture of the e-puck2 at nine centimetres from the face. Even though the photograph used for each scenario was dark, the algorithm is sufficiently trained to detect the face in these conditions.

5.7 Results

The following graphs are time averages in seconds for each different scenario. Each of them has approximately thirty to fifty samples to compute their average. The samples are dependent on the time it takes for the robot to do a loop. Therefore, the slower is the loop, the smaller the amount of samples will be for a scenario.

Figure 44 illustrates the average results and figure 45 shows the box plots of the first benchmark. This performance computes the time taken to take a picture of the face and save it in the computer disk that will process the face detection algorithm. These figures shows that the fastest scenario is the one that uses WiFi to control the robot with the small resolution image. The scenarios that send the images to the PC performed the worst and the scenario that used the scp command with the high resolution camera is the one that took the longest time.

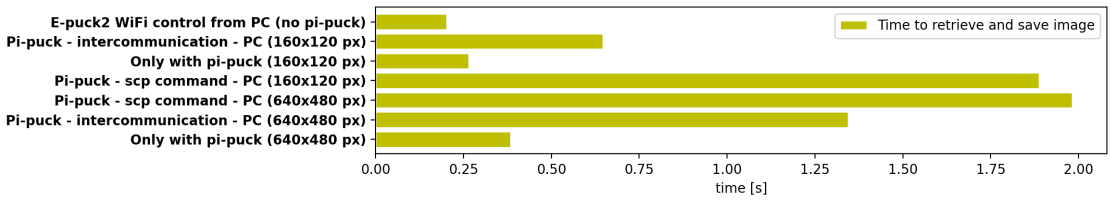

Figure 44: Time to take and save a picture

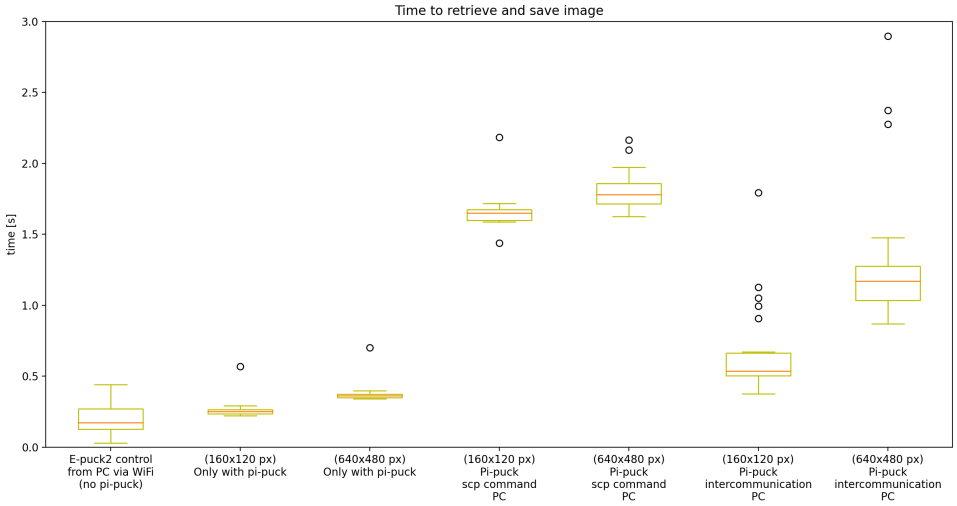

Figure 45: Box plots of the time to take and save a picture

Figure 46 shows the average results and figure 47 shows the box plots of the second computed performance. As already mentioned, the second benchmark is the time it takes for the computer to process the face detection algorithm and instruct the robot which direction it should go.

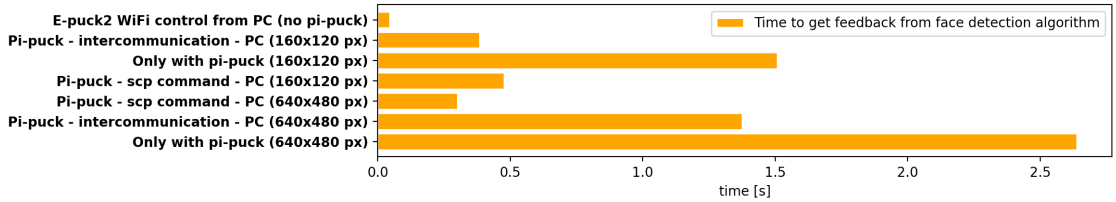

Figure 46: Time to process and get feedback of an image

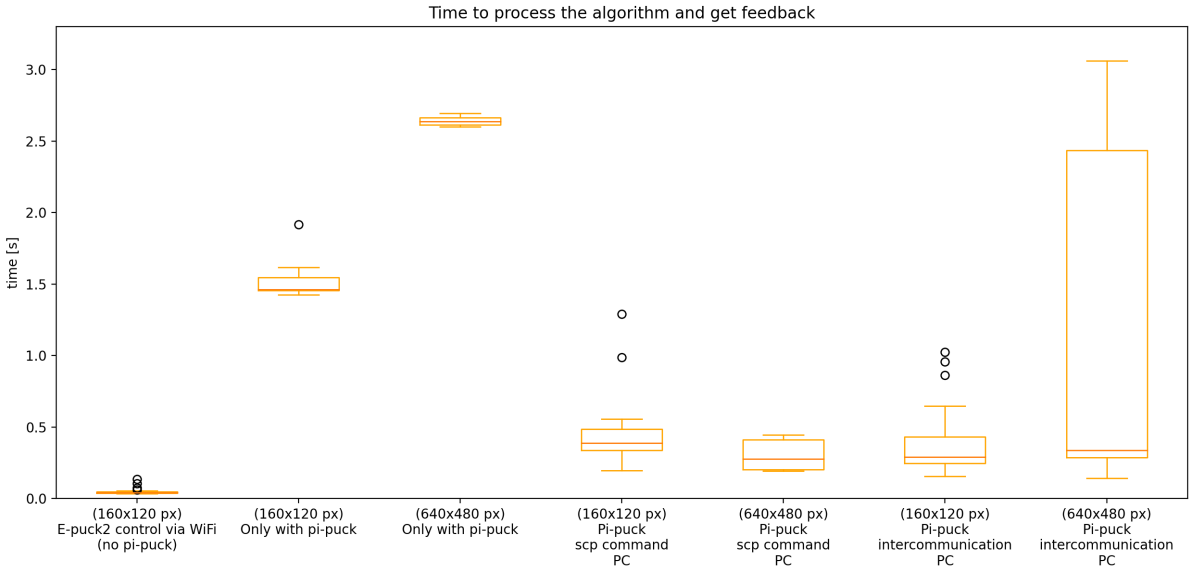

Figure 47: Box plots of the time to process and get feedback of an image

The scenarios that used the MacbookPro (PC) were the fastest. These scenarios could be improved upon to gain precious deci-seconds if the intercommunication were better. As for the Raspebrry Pi scenarios, where it runs as a stand-alone, it is slower than the standard PC and both are in the bottom position for these benchmarks because of the heavy face detection algorithm.

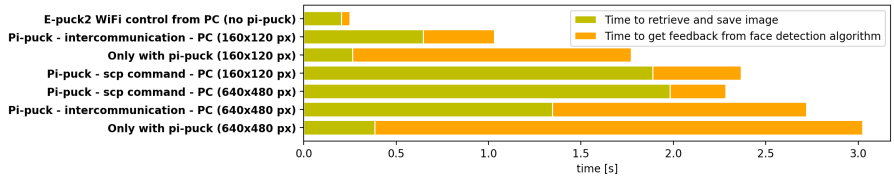

Figure 48: Sum of the results of figure 44 and 46

Figure 48 shows the sum of figures 44 and 46. Although the WiFI scenario is by far the fastest, the result might have been different if the PC had to control many robots while processing their images. One could imagine that this would give the Raspberry Pi an advantage as it would be stand-alone and the intercommunication can be overwhelmed.

Conclusion

In order to carry out the project of developing a new API, a communication protocol via WiFi with e-puck2 was written in Python. Once the WiFi intracommunication was working, the Webots protocol was added. After developing these two platforms, they had to be combined so that the user could write a single controller for the robot. The intercommunication between the e-pucks was written so that the robots can communicate independently of the platform. The development of the communication led to programming a graphical user interface to control and enjoy the robot's point of vision. Once the code was ready, the package had to have documentation and be easy to download on the web so that anyone could access the API. While the API was being tested by students, the pi-puck interface was written and then combined with the two other platforms.

As a result of the work done for this report, the e-puck2 has become even easier to use for programmers. It can now be controlled on three different platforms (WiFi, simulation, pi-puck) using a single controller. The robot works well with the pi-puck for basic tasks but still needs some improvements. For example, the computational speed can easily be improved by replacing the Raspberry Pi with a more powerful one and the speaker can be improved if the configurations are done correctly on the Raspberry Pi. The pi-puck does not show many beneficial aspects when only one robot is being used and it would be really interesting to try using several robots at the same time. Going forward, the unifr_api_epuck package will now make possible many new and exciting projects thanks to Python. It is available for free and for everyone. It is expected that students at the University of Fribourg will use it during the next few years and have fun writing in Python.

References

[1] Mark Bustos. *'Spot' the Robot Dog Tested in Combat Scenarios to Prepare for Military Applications, Battlefield Robotization*. Apr. 2021. URL: https://www.sciencetimes.com/articles/30595/20210411/french-army-tests-spot-robot-dog-military-applications.htm.

[2] Espressif Systems Documentation. *ESP32 - Wi-Fi Driver*. URL: https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-guides/wifi.html#esp32-wi-fi-throughput.

[3] EPFL. *e-puck2 description with image*. URL: http://www.e-puck.org/.

[4] Anna Förster. *e-puck1 description with image*. URL: https://www.researchgate.net/figure/The-e-puck-robot-with-its-components-It-is-a-small-but-complex-robot-with-a-lot-of_fig2_319405680.

[5] *robot*. URL: https://dictionary.cambridge.org/dictionary/english/robot.

[6] James Vincent. *The NYPD is sending its controversial robot dog back to the pound*. Apr. 2021. URL: https://www.theverge.com/2021/4/29/22409559/nypd-robot-dog-digidog-boston-dynamics-contract-terminated.