Robocologie User Manual

Table of contents

I. Network

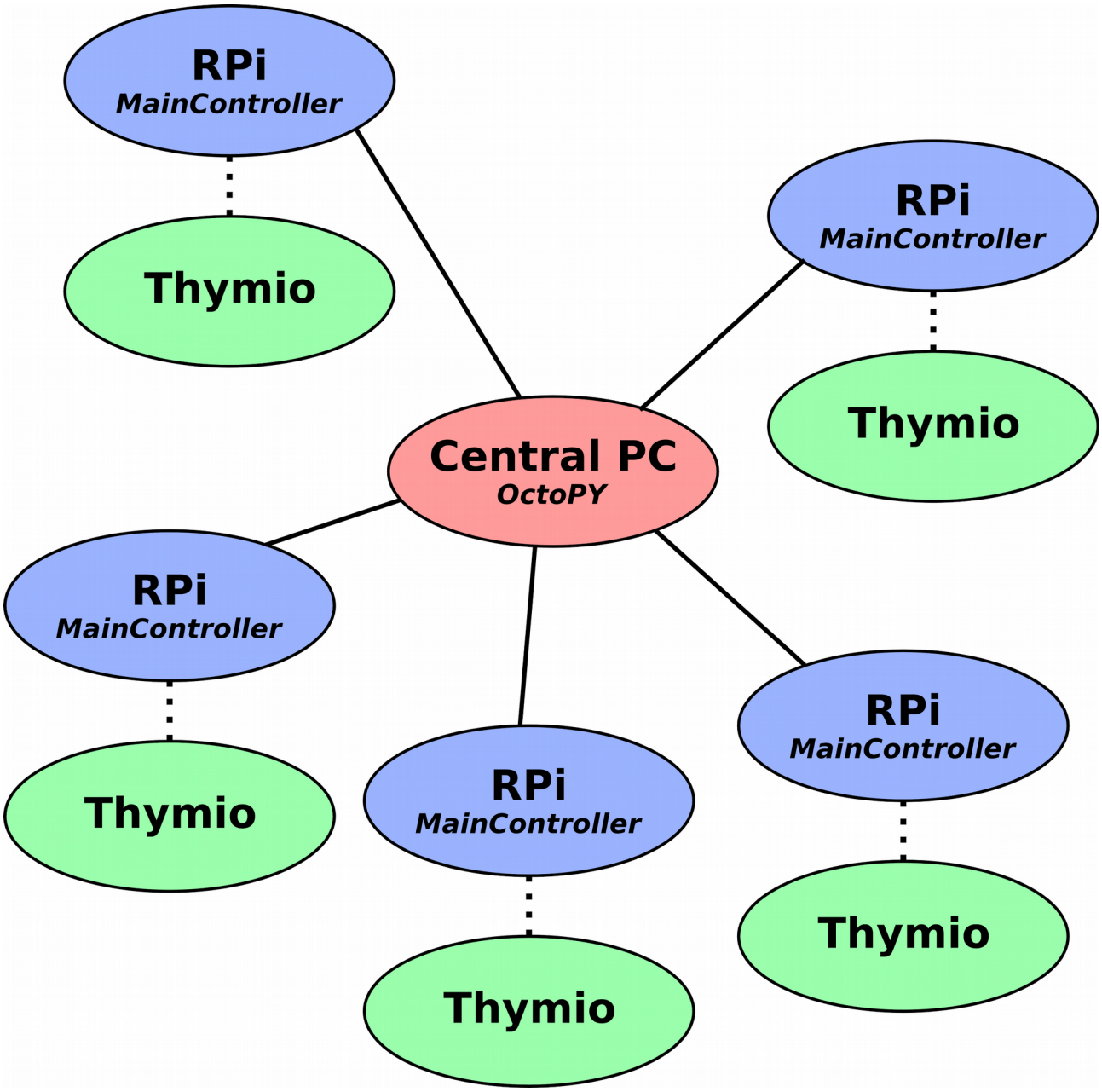

Illustration 1: Network in Octopy, at the Hardware level.

Central PC:

The central PC is used to communicate with every raspberry pi by using the OctoPY software. Communication is achieved with WiFi by a private network that is managed by the central

PC.

As such, is it necessary at first to launch this network so that raspberries pi can connect. This is done by having the central PC connect to the access point. The file **/etc/network/interfaces** is configured so that the PC can easily be connected to the access point. To connect to the access point, type the following commands:

```
~$ sudo ifdown eth1 // Not necessary but might be needed when the connection has not previously been properly cut
~$ sudo ifup eth1 // This may display errors but they are not blocking
~$ ping 192.168.0.100 // This is used to confirm that the PC can communicate with the access point
~$ ifconfig // It is also safer to validate that the adress of the PC is correct: 192.168.0.210
```

When the PC is correctly connected to the access point, it is necessary to start the DHCP server so that raspberries can automatically be issued IP addresses:

```
~$ sudo /etc/init.d/isc-dhcp-server stop // Again may not be necessary but is more cautious
~$ sudo /etc/init.d/isc-dhcp-server start
```

Alternatively you can run the script *net_conf.sh* that runs all commands above. It's located at *./thymioPYPI/OctoPY/net_conf.sh*.

```
~$ bash net_conf.sh
```

The password will be asked and you will only have to stop the ping command when the router responds.

Raspberry Pi:

Raspberries are configured so that they can automatically connect to the network at launch. IP addresses are issued between 192.168.0.110 and 192.168.0.150. If an error occurs, it is possible to manually connect to the network by using **wicd-curses**, which is already installed on the raspberries:

```
~$ wicd-curses
```

Here are the network information:

- **SSID:** NETGEAR_11ng
- **Protocol:** WPA
- **Passphrase:** rpiaccesspoint

II. OctoPY

The software used to remotely control the robots is called OctoPY. OctoPY is mainly used to send basic instructions (start, stop etc...) to the raspberries. The software can be found at *thymioPYPI/OctoPY/OctoPY.py*.

You can launch OctoPY as follow :

```
~$ python OctoPY.py [-h] // The -h argument will show all others arguments available
```

Commands:

Look:

Usage: look [range] [-s save_table]

This command is used to look for the raspberries connected on the network. The default range of IPs on which to look is 192.168.0.111-150 but another range can be given through the *range* argument. If the argument *-s* is given, when a raspberry is found, its hostname is retrieved. The association between IP addresses and hostnames is then saved in the hostnames table. This table is saved in a file named *hostnames_table.json* in the same directory. When OctoPY is started, if this file exists, the hostnames table is loaded so that no **look** command is necessary if they was no changes in the raspberries connected to the network.

Send:

Usage: send <message> [hosts_list] [data]

This is the main command that you will use during your experiments. This command is used to send instructions to particular robots (by specifying the *hosts_list* argument) or every known robot. Additionally, the argument *data* can be used to send any necessary data with the instruction.

Messages:

Instructions are sent by specifying a particular message. This message is specified known by its numerical ID, the list of which can also be displayed in OctoPY by taping the TAB key of the keyboard after *send* has been written. Here is the list of the current messages:

- **0 : INIT:** This instruction must always be sent to the raspberries before other instructions can be specified. This is used to launch the main controller on the robots as

well as *aseba*. Any other instruction save **OFF** will fail on a non-initialized raspberry.

- **1 : START:** Launches the experiment specified by the configuration filed loaded on the raspberry. If no configuration file has be previously loaded with the **LOAD** message then a default experiment will be launched (a simple braitenberg).
- **2 : PAUSE:** Pauses a started experiment on the raspberry.
- **3 : RESTAT:** Restarts a paused experiment.
- **4 : STOP:** Stops the current started experiment.
- **5 : KILL:** Stops the controller on the raspberry. For safety, also stop the current started experiment. **INIT** must be sent again before any other messages can be received by the raspberry (save **OFF**).
- **6 : OFF:** Shutdowns the raspberry.
- **7 : SET:** Sets an experiment on the raspberry. The first argument must be configuration file (.cfg file) of the experiment to set then the address(es) of the raspberry(ies).
- **8 : DATA:** Used to send any data to a raspberry.

Hosts list:

The list of raspberries to which the message must be sent can either be precised as an IP address or as a hostname if this hostname exists in the hostnames table (see **look** command). TAB can be used to display the list of possible hostnames.

Query:

Usage: query [hosts_list]

Queries the state of the raspberry. This state can be varied:

- **Down:** The raspberry is off or not connected to the network.
- **Sleeping:** The raspberry is on and connected to the network but not initialized by the **INIT** command.
- **Listening:** The raspberry is initialized.
- **Started:** An experiment is started on the raspberry.

State:

Usage: state [hosts_list]

Gives the state (as defined previously) of a list of raspberries according to the a previously

executed **query** command. If no **query** has already been executed then a **query** is done.

Launch:

Usage: launch controller_config_file [-d detached]

Launches a Controller according to its configuration file (.cfg file). When the controller is launched, no other commands can be used.

If the argument *-d* is given, the controller is detached which means it will run in the background (allowing multiple controllers to run at the same time) while OctoPY can still be interacted with. If *-d* is not given, a controller can be stopped while executing *step()* with CTRL+C and will execute *postActions()* before stopping (please see the controller dedicated section to learn what to do in *step* and *postActions*).

Put:

Usage: put <src_path> <absolute_dest_path> [hosts list]

Use this command if you need to send file(s) from the server to particular raspberries (by specifying the *hosts_list* argument) or every known raspberries. It uses scp so you can specify similar path arguments.

Get:

Usage: get <absolute_src_path> <dest_folder> [-r] [hosts list]

Symetric of the previous **put** command. Use it if you need to get file(s) located at the same place on particular raspberries (by specifying the *hosts_list* argument) or every known raspberries and send them to the server. Destination on server must be an existing folder so that sub-folders for each host are created inside with an appropriate name. If the argument -r is given, successfully sent files will be removed from hosts.

Help:

Usage: help <command_name>

At any time in OctoPY, you can use help to display a short help message about the command given in argument.

Simple Scenario:

This last section is used to illustrate a simple succession of basic instruction in OctoPY, in

this case to launch the experiment *SimulationBraitenberg*.

```
>> look -s // Gathers the addresses and hostnames of raspberries connected on the network

// If you changed something in the configuration file, send it to every raspberry

>>put rpifiles/experiments/braitenberg.cfg ~/dev/thymioPYPI/OctoPY/rpifiles/experiments

>> send 0 // Initialize all the raspberries

>> set braitenberg.cfg // Loads the experiment on every raspberry, same as “send 7 braitenberg.cfg”

>> send 1 pi2no02 // Starts the experiment on the raspberry pi2no02 only

>> send 1 // Starts every raspberry

>> send 4 // Stops every raspberry

>> send 5 pi2no02 // Kills the controller on raspberry pi2no02

>> send 6 pi2no02 // Shuts the raspberry pi2no02 off
```

Monitoring the raspberries:

While it is not necessary to directly connect on the raspberries, it is often useful to check on the execution of the program. First connect to the raspberry by ssh:

```
~$ ssh pi@<IP address>
```

The username is thus *pi* and the password *pi* also. Then go the folder where the log files are written:

```
~$ cd ~/dev/thymioPYPI/OctoPY/rpifiles/log
```

Then you can read the log file as it is written by typing the following command:

```
~$ tail -F MainController.log
```

III. Experiments

Experiments are the programs launched on the raspberries by the OctoPY and constitute the specific behaviours desired for the robots. OctoPY comes with a framework that is intended to facilitate the integration of new experiments. As such, a minimal implementation of an experiment corresponds to coding the behaviour of the robot at each step of the experiment. The code necessary for experiments is in the *OctoPY/rpifiles/experiments* folder. On the raspberries, the corresponding code can be found at *~/dev/thymioPYPI/OctoPY/rpifiles/experiments*. Several experiments already exist to serve as easy demonstrations on how to code an experiment. It is advised to create the experiments on the Central PC and then copy them (with the **put** command) on every raspberry.

Creating an experiment:

A script exists to quickly create the files necessary to code an experiment: ./thymioPYPI/OctoPY/rpifiles/*CreateExperiment.py.*

Usage: python CreateExperiment.py experiment_name

This script creates the basic files and folders that will be used for the experiment *experiment_name*. More precisely, the following files are created:

- **./experiments/config_*experiment_name.cfg*:** This file is used to specify the path to the folder and main source file of the experiment which should not be changed. But this file can also be used to specify any parameter that we desire to load with the experiment. Each parameter must be written as follows : *type parameter_name = parameter_value*

Three different *types* are recognized when parsing this file: *int*, *float* or *str*. If no *type* is given, then *str* is assumed. Any parameter given in the configuration file may then be accessed in an experiment by using *Params.params.parameter_name*.

- **./experiments/experiment_name :** This folder contains all the other files created for the experiment. This folder acts as a Python module which is then used by the main controller of the experiments.
- **./experiments/experiment_name/__init__.py:** This empty file should often not be changed as it is only used so that the folder is recognized as a Python module.
- **./experiments/experiment_name/readme.txt:** This is self explanatory. This file should be used to clearly describe what is the purpose of the experiment.
- **./experiments/experiment_name/SimulationExperiment_name.py:** This is the main file where the code will be written. The next section will describe more in the details the

content of this file.

The simulation file:

As previously stated, this file is where the major part of the code will be written (in the first time). When created, its content is at follows:

```
import Simulation
import Params

class Simulationexperiment_name (Simulation.Simulation) :
	def __init__(self, controller, mainLogger) :
		Simulation.Simulation.__init__(self, controller, mainLogger)

	def preActions(self) :
		pass

	def postActions(self) :
		self.tController.writeMotorsSpeedRequest([0, 0])

	def step(self) :
		pass
```

In this file is defined the experiment class which extends for the Simulation class defined in the framework. It overrides three functions from its parent class: *step*, *preActions* and *postActions*.

step is the most important function to write. When an experiment is started, *step* is the function called at each step of the experiment. In consequence, this is where most of the code will be called. p*reActions* is used to define all the instructions that must take place when the experiment is started but before the first call of the *step* function. Finally *postActions* is used to code the instructions taking place before the experiment is stopped with send 4 or when an error occurs an make the simulation crashes.

Tools:

To help you coding your experiment, there are some small APIs that already exist and that you can import into your simulation. These APIs are located in ./thymioPYPI/OctoPY/rpifiles/tools.

For now (11/05/2017) there is only one API you can use to facilitate your work with the camera : camera_tools. You may read the README.txt and import it as following in your simulation file :

from tools.camera_tools import <class you need>

Sending instructions to the Thymio:

Sending instructions to the Thymio is easy as an interface (in the file *ThymioController.py*) is used to transfer the most basic instructions to the Thymio through *aseba*. The *self.tController* property is used to access this controller and interact with the Thymio from your simulation file. These are the instructions that currently exists in the controller:

- **readSensorsRequest:** Reads the values of the proximity sensors. These values must then be accessed thanks to the *getPSValues()* function of *tController*. It returns an array of 7 values in [0, ~ 4300]. 4300 meaning the robot is against a wall.
- **readGroundSensorsRequest:** Reads the values of the ground sensors. Accessed thanks to the *getGroundSensorsValues()* function. Return format is a tuple: ([a1, a2], [r1,r2], [d1, d2]). a1 and a2 are in {0, 1} and the others values are in {0, …, 1023}.
- **readMotorsSpeedRequest:** Reads the speed values of both motors. Accessed thanks to the *getMotorSpeed()* function. It returns an array of 2 values in [-500, 500] (500 = ~ 20 cm/s).
- **readAccRequest:** Reads the accelerometer values. Accessed thanks to the getAccValues() function. It returns an array of 3 values in {-32,-31, …, 31, 32} (23 = ~ 1g)
- **writeMotorsSpeedRequest:** Sends the instruction to move to the robot. The desired speed of each motor is specified as an argument under the form of an array: *[left_motor, right_motor]*. Each value must be in [-500, 500] (500 = ~ 20 cm/s).
- **writeColorRequest:** Sends the instruction to change the color of the robot's LED. The desired color is specified as an argument under the form of an array: *[R, G, B]*. Each value must be in {0,1, …, 32}.
- **writeSoundRequest:** Sends the instruction to emit a sound. The desired sound is specified as an argument under the form of an array: *[frequency in Hz, duration in s]*.

The Thymio updates its values at a different frequency for each variable. Moreover, your simulation will probably not be synchronized with any of those frequencies, so you may want to know whether your getting a new value or not when calling any *getXXXX()* function. This may be useful if you need to initialize variable in your *preActions()* function .

To do so, the function **isNewValue(varName)** is available and will return true if the value of *varName* has been updated by the Thymio since the last time you called the *getXXXX()* function associated to *varName*, false otherwise. *VarName* must be a String in {“MotorSpeed", "PSValues",

"GroundSensorsValues", "AccValues"}. If used, this function must always be called before the *getXXXX()* function you'r testing :

```
“ ” ”
Example 1 : user want to be sure that accelerometer values have been updated since the last time he asked
“ ” ”
self.tController.readAccRequest()
new = self.tController.isNewValue(“AccValues”)
accValues = self.tController.getAccValues()
if new:
        # user got new values (Thymio has updated them)
else :
        # user got previous values (Thymio has not updated them yet)
```

```
“ ” ”
Example 2 : user wants to initialize accelerometer values in preAction()
“ ” ”
new = False
while not new:
        self.tController.readAccRequest()
        new = self.tController.isNewValue(“AccValues”)
        time.sleep(0.1) # avoid flooding thymio since its frequency is ~ 10Hz
accValues = self.tController.getAccValues()
```

The document https://aseba.wdfiles.com/local--files/fr:asebausermanual/ThymioCheatSheet_fr.pdf shows everything you need to know about the Thymio and even more.

Communicating with other robots:

Robots can communicate with each other by using the **sendMessage** from the parent *Simulation* class.

Usage: self.sendMessage(recipients = recipientsList, value = value)

The *recipientsList* corresponds either to a list of IP addresses or hostnames as when using OctoPY.

To receive a message, the function **receiveComMessage** must be overrided.

Usage: def receiveComMessage(self, data)

data[“senderHostname”] contains the sender hostame *data[“value”]* contains the message's value.

IV. Controllers

Controllers are used to automate the instructions that are sent by OctoPY so that it not necessary to manually type them. As for experiments, controllers can be created with a script called *CreateController.py* (*./OctoPY/CreateController.py*). All controllers are located in ./thymioPYPI/OctoPY/controllers. And again as for experiments, this script will create a configuration file (.cfg file) and folder containing a file with basic code for a controller :

```
import Controller
import Params

class ControllerName(Controller.Controller) :
	def __init__(self, controller, mainLogger) :
		Controller.Controller.__init__(self, controller, mainLogger)

	def preActions(self) :
		pass

	def postActions(self) :
		pass

	def step(self) :
		pass

	def notify(self, **params) :
		pass
```

Similarly to experiments, there are three main functions to override: *preActions*, *step*, and *postActions*. By using the property *self.OctoPYInstance*, it is possible to use OctoPY commands in

the same way as with the interactive version. Typically, *preActions* will be used to initialize and start experiments, *postActions* to stop the experiments and shut the raspberries down and *step* to code the behaviour of the controller while experiments are running.

The most useful behaviour of a controller while experiments are running is to listen to the raspberries. This can be done by *registering* the controller to particular raspberries. This is done with the function **register**. Then, overriding the function **notify** allows to code the behaviour of the controller when notifications are received from the raspberries listened to. You can check the controller **TestNotifications** to have an example on how to code such a controller.

V.Camera Tools

In order to use the camera module in your experiment (Section III) extend the camera_tools.Image_Processor.Detector class to create your detector.
For camera settings modify the camera_tools.py/settings.py. Available settings are descibed in the PiCamera Documentation linked in the file.

Extending the Detector Class to create a Detector.

There are 2 way to do this:

- Create the scaffold for Your_Detector extending the Detector.
- In your __init__ method use the super constructor with a call to Detector.__init__(self)
- Implement the post_processing_function(self,pre_processing_output) method (#TODO) the second argument contains an RGB_image provided by the camera module implemented as a 3D numpy array each dimension a channel. The return value of this method can be fetched by calling the method Your_Detector.get_results() with a prepended flag new_results (see below Detector Class methods description)

```
import camera_tools
from camera_tools import Image_Processor.Detector as Detector

class Your_Detector(Detector):
	def __init__(self):
		Detector.__init__(self)

	def post_processing_function(self, pre_processing_output):
		#TODO
```

Second way:

- You can also implement your own pre_processing_function(self, RGB_image) the second argument contains an RGB_image provided by the camera module implemented as a 3D numpy array and the returned value is piped to the post_processing_function.

```
...
        def pre_processing_function(self, RGB_image):
                #TODO
                return pre_processing_output
...
```

An example: EdgeDetector Class.

This example is given as a guideline for implementing Detectors.

```
import camera_tools
from camera_tools import Image_Processor.Detector as Detector
from camera_tools import image_utils as image_utils

class (Detector):
        def __init__(self):
                Detector.__init__(self)
                # using the setter method is equivalent to implementing
                # the method pre_processing_function
                self.set_pre_processing_function(image_utils.convert_to_HSV)

        def post_processing_function(self, pre_processing_output):
                gray_image = pre_processing_output[:,:,2]
                edge_image = image_utils.automatic_canny(gray_image)
                return edge_image
```

Notice: self is passed to the function applyed to the frames in the stream provided by the camera module, this allow to update the state of the class and store informations in the detector.

Include YourDetector in the Experiment Lifecycle

In order to include a detector in the experiment it must respect the experiment ltfecycle.
In other words methods calls are intended to follow this order.

Simulation	**Detector**
__init__(...)	YourDetector(...), *constructor*
preActions(...)	start()
postActions(...)	shutdown()
step(...)	get_results()

A clarificatory example is provided, the previous table might be useful as a reminder.
This example assumes YourDetector is a python file in the same directory of the Simulation file.

```
from YourDetector import YourDetector
class YourSimulation(Simulation.Simulation) :
	def __init__(self, controller, mainLogger):
		self.your_detector = YourDetector()
	def preAcrions(self):
		self.your_detector.start()
	def postActions(self):
		self.your_detector.shutdown()
	def step(self):
		new_results, post_processing_output = self.your_detector.get_results()
		... Check Detector Class methods and description to handle outputs
```

Detector Class methods description

Python do not explicitely provide syntatic support for interfaces but it can be implemented with a class by raising NotImplementedError in a non overloaded method.
This class requires some methods to be implemented.

start():

start the camera module and frames stream.

shutdown():

- shutdown the camera module gracefully.

get_results():

- a non blocking request to the Image Processor for new post_processing_output.
-
- returns the tuple (new_results, post_processing_output)
- returned values define 3 possible states:
    - *post_processing_output == None*. state: not ready.
        - the function was called before any possible processing could be done on the captured image, putting sleep at the top of your experiment to allow camera to warm up and processing.
    - *new_results == Fals*e, state: old results fetched.
        - If the post_processing_function is computationally heavy this may frequently occour as the caller runs in another thread.
    - new_resuts == True, state, new results fertched.
        - post_processing_output will contain new results from the latest image caputerd.

post_processing_function()

This function has to be overloaded by the class extending the Detector Class, if not NotImplementedError will be raised upon call.

set_pre_processing_function(self,pre_processing_function):

This method sets the preprocessing function of the ImageProcessor

Use the TagDetector

To use the tag detector the main steps are in the display,
notice: a standalone script is provided in *camera_tools/standalone_test.py*

```
from camera_tools import TagDetector as TagDetector
...
tag_detector = TagDetector()
tag_detector.start()
while(<running exactuion loop>):
    new_results, tags_info = tag_detector.get_results()
    if new_results:
        tags_countours, tags_ids, tags_distances, tags_rotations = tags_info
        ... use the results ...
tag_detector.shutdown()
```

```
…
```

Generate tags

To generate tags use the script camera_tools/genetate_tags.py
usage: python generate_tags.py [all] : will generate 512 tags, 6 per file in svg format
usage: python generate_tags.py [<number>] : will generate <nuber> tag, in svg format

VI. Raspberry

Most of the raspberries are already configured. But it may be necessary to set new raspberries when the time comes. Fortunately, this is easy.

There are two raspberry images stored on the computer :

- *~/rpiImage_pi3noXX_2016_12_15.img* (Wheezy version of Debian)
- *~/rpiImage_pi3no06_2017_06_08.img* (Jessie version of Debian)

Update a Raspberry

If you want to update a raspberry, first connect it to the Internet. The easiest way to do so is to directly connect the raspberry to a screen, indeed ssh connection is not very convenient as the ip address will change when connecting to the Internet. Then type :

```
~$ wicd-curses
```

and choose a wireless connection. Once connected type :

```
~$ sudo apt-get update
~$ sudo apt-get dist-upgrade
```

The second one informs you of how much space it will take so first check if there is enough space left (*df -h* will tell you).

Finally type :

```
~$ sudo apt-get -f install
```

that will resolve dependencies and will tell you about problems (if any).

Create an image of a Raspberry

If you want to create an image of a raspberry (used for backup or to spread to other raspberries) take the SD card of the raspberry out and connect it to the computer.

Then find the devices mounted on the computer with:

```
~$ df -h
```

and find which one corresponds to your SD card (it should be something like /dev/sdb1 and /dev/sdb2 if it has two partitions).

Then, and regardless of the number of partitions type :

```
~$ sudo dd if=/dev/sdb of=path_image_created/name_image_created
```

Note that *pi* is not allowed to use sudo, so first type :

```
~$ su nicolas
```

Please ask Nicolas Bredeche for the password.

Set up a Raspberry

First Method

The first step is to write the default system image on the SD card. The newest images is present on the computer at: *~/rpiImage_pi3noXX_2016_12_15.img*

To write this image on the SD card, first find the devices mounted on the computer:

```
~$ df -h
```

Then, find the partition on which the SD card is mounted. It should be listed as something as */dev/sdc1* and */dev/sdc2*. You can unmount the device by typing the following command:

```
~$ umount /dev/sdc1 // If /dev/sdc1 is the partition on which the SD is mounted
```

You can then copy the system image on the SD by using the following command:

```
~$ dcfldd bs=4M if=<system_image> of=<partition>
```

This should take some time. If you need more information, please read https://www.raspberrypi.org/documentation/installation/installing-images/linux.md

Second Method

The second method is faster as you just have to launch *Etcher,* then you select the latest image present on the computer at *~/rpiImage_pi3no06_2017_06_08.img* and select the storage device you want to write the image on. Finally, you press Flash. It takes about 30 minutes to flash.

As this image is copied to every raspberry, there are some changes that necessary to do so that each raspberry can be unique. First it is necessary to change the hostname of the raspberry. Each raspberry hostname should read as follows: **pi<raspberry_version>no<raspberry_number>**. For example, for the third raspberry pi 2, its hostname is **pi2no03**. Whatever the raspberry's version (2 or 3), please keep the raspberry numbers successive. To change the hostname of a raspberry, first modify the file */etc/hosts*. Find the line that reads:

```
127.0.1.1          raspberrypi
```

and change *raspberrypi* by the new hostname. Then open the file */etc/hostname* and replace the hostname by this new hostname.

You also need to change the machine-id so that each agent as an unique one. To do that, type the following command:

```
~$ dbus-uuidgen > /var/lib/dbus/machine-id
```

The system image may not have the last version of the code available on the git repository. As such, you need to pull the last version of the code from the git. There is a basic git account created to pull (and not push) code from the git. Its credentials are:

- **Username:** ThymioPYPI
- **Password:** xxxxx (please ask Nicolas Bredeche)

Then, to pull the code move into the *~/dev/thymioPYPI* directory and type the following command:

```
$~ git pull origin master
```

Alternatively, you can use the put (described earlier) command to update the version of code without having access to the internet.

If you have an ssh error while trying to send messages to the robot with OctoPY, you may solve it by connecting to the raspberry with ssh -X pi@192.168.0.XXX (if it is the first time you connect to it, a key has to be generated).

Any question?

If you encounter any problem, you can check the FAQ Raspberry/Thymio (appendice 3), some answers may be deprecated.

VII. Files Location

This section is used to summarize the location of the different files and folders on the thymioPYPI repository:

- ***thymioPYPI/OctoPY*** contains all the files used by the OctoPY framework
  - ***./controllers***: contains the controllers and their configurations files
  - ***./log***: contains a log of all commands typed in OctoPY when launched with the -L argument
  - ***./rpifiles***: folder containing all the files that are supposed to be used on the raspberries
    - ***./experiments***: contains the experiments and their configurations files
    - ***./filesNetworking***: folder containing a copy of all the files that are used to configure the network.
    - **./log**: contains the log of the part of OctoPY that runs on the raspberry as well as the experiments log.
    - **./standAlone**: projects or pieces of code that run independently of OctoPY (they are not experiments so you need to manually launch them from a ssh connexion)
    - **./tools**: contains small APIs that can be used in your experiments

  Every other folder contains project from students who have worked with the Thymios (but not necessarily using OctoPY)

  - ***thymioPYPI/p_androide-SOTO_SHAMS****:* M1 ANDROIDE project. They upgraded OctoPY and used it to implement Embodied and Distributed evolutionary algortihms (experiments FollowLightGen and FollowLightGenOnline)
  - ***thymioPYPI/robocologieimage***: source code for the tracking and monitoring software.
  - ***thymioPYPI/RFID***: simple test code to use the bar scanners.
  - ***thymioPYPI/TagRecognition***: source code and data for tag recognition. Most of the useful code is also present in thymioPYPI/OctoPY/rpifiles/standAlone/TagRecognition.

VIII. Manual History

This section keeps history of every modification of the manual.

- 05/02/2017 : 'put' and 'get' commands added at the end of the section *Commands* of **II. OctoPY**. Paragraph of introduction in **III. Experiment** changed accordingly.
- 06/02/2017 : instructions to run the *net_conf.sh* script added in the section *Central PC* of **I. Network.**
- 20/03/2017 : instructions to get accelerometer values added in the section S*ending instructions to the Thymio* of **III. Experiments.**
- 07/05/2017 : instructions to stop a controller added in **II. OctoPY** → Commands → Launch
- 09/05/2017 : instructions send 5 updated in **II. OctoPY** → Commands → send + examples added in S*endings instruction to the Thymio* in **III. Experiments**
- 11/05/2017 : **VI. Files locations** updated, **VIII. Appendices** added, **III. Experiments** → Tools added
- 18/05/2017 : **VI. Files locations** updated to **VII**, **VIII. Appendices** updeted to **IX**, **V**

  Camera Tools → added
- 09/06/2017 : **VI. Setting a Raspberry** updatd to **VI. Raspberry** and added three parts : **update a raspberry**, **create an image of a rasberry** and **set up a raspberry.** Also added **Appendice 3 : FAQ Raspberry/Thymio**.
- 15/06/2017 : Added how to update from wheezy to Jessie on a raspberry in **Appendice 3 : FAQ Raspberry/Thymio**.

IX. Appendices

Appendice 1: OctoPY diagram

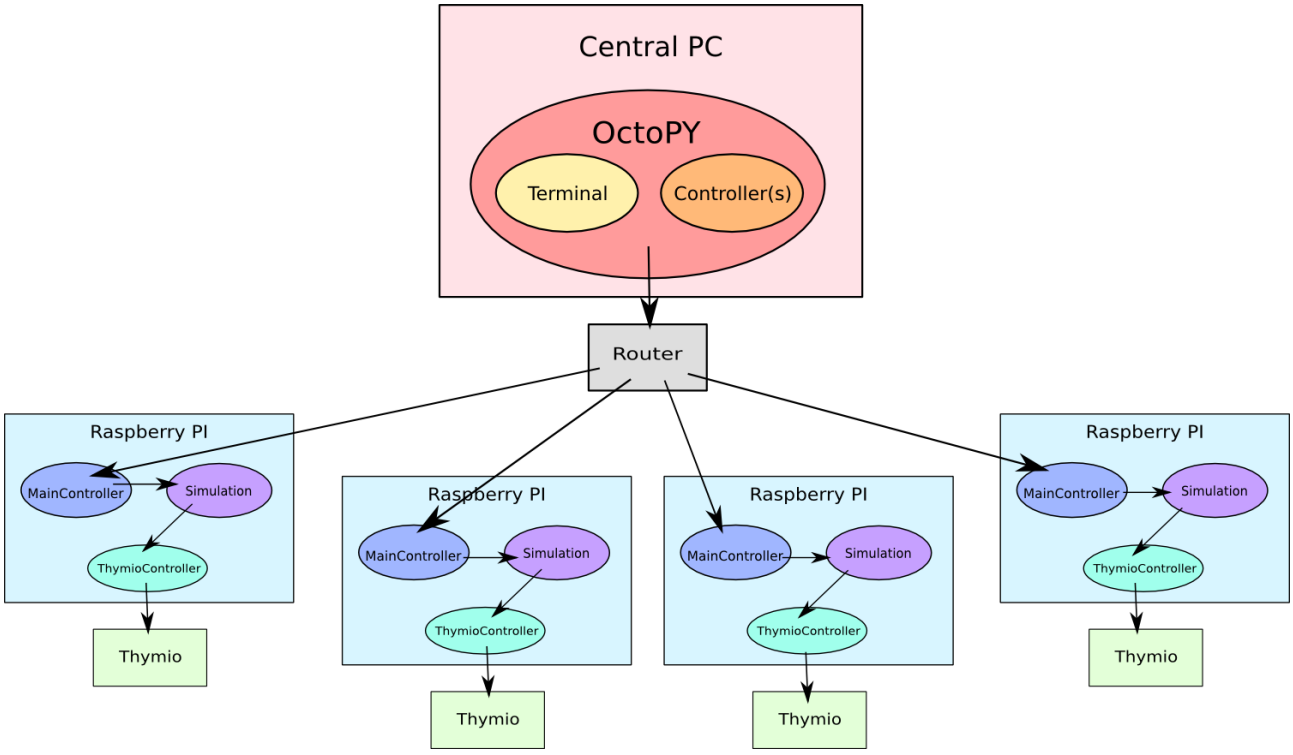

Illustration 2: Full diagram of the OctoPY framework

Appendice 2: In case of emergency

If your having big troubles when trying to start a simulation and you already tried to send 5 and send 0 again you can either reboot the raspberry or try the following sequence :

```
~$ ssh pi@<IP address> // connect to the raspberry that causes the problems
~ $ ps -aux
// In the list, look for the PIDs of process whose owner is PI and corresponding to :
// – MainController.py
// – Asebamedulla
// – Any ssh connection that is not the one you are using right know
~ $ kill <list of PIDs you just determined>
```

Appendice 3: FAQ Raspberry/Thymio

Note that some answers may be deprecated.

Q: RaspPi 2 default login/password with raspbian?

A: pi/raspberry -- to change that to whatever: sudo passwd pi

Q: How to update a Raspberry from Debian Wheezy to Debian Jessie?

A: First type *cat /etc/os-release* to know which version of Debian you have.
When you checked your version of Debian type :

```
sudo apt-get update
sudo apt-get upgrade
sudo apt-get dist-upgrade
```

Depending on the raspberry this could take a while. Also note that you need an internet connection to do that.

Then you need to replace the Wheezy sources by Jessie sources :

```
sudo sed -i /deb/s/wheezy/jessie/g /etc/apt/sources.list
sudo sed -i /deb/s/wheezy/jessie/g /etc/apt/sources.list.d/*.list
```

Finally type the first three commands again :

```
sudo apt-get update
sudo apt-get upgrade
sudo apt-get dist-upgrade
```

At this stage, your version of Debian should be Jessie, you can check it by typing *cat /etc/os-release*

source : `http://www.journaldulapin.com/2016/03/04/passer-de-wheezy-a-jessie-sur-un-raspberry-pi/`

Q: Keyboard is english/french layout, I want the other way around.
A: setxkbmap us (or: setxkbmap fr)

Q: GDBus.Error:org.freedesktop.
A: http://raspberrypi. : Open LXTerminal ; Type lxsession-edit ; uncheck LXPolKit ; OK ; reboot

Q: How to set up Aseba for Raspberry PI
A:
download aseba: https://aseba.wikidot.com/en:
sudo dpkg -i aseba_1.5.5_armhf.deb
sudo apt-get -f install

Q: How do I connect a Thymio?
A:
(1) connect Thymio ; lsusb (should see: "Ecole Polytechnique ..."
(2) dmesg | tail ; ls /dev/ttyACMx (x is replaced by the number from the dmesg msg)
note that some USB cable are power-only enabled (ie. no data)

Q: I see a thymio in Aseba, but cant connect

A: sudo adduser $USER dialout

Q: a simple example for using Thymio with RaspPI in Python
A: https://aseba.wikidot.com/en: (obstacle follower)

Q: useful packages?
A:
sudo apt-get install subversion
sudo apt-get install lynx
sudo apt-get install vlc
sudo apt-get install wicd
sudo apt-get install wicd-curses

Q: useful applications?
A:
leafpad (X) -- text editing
wicd-gtk (X) and wicd-curses (terminal) -- wifi and ethernet
vlc
gpicview (X) -- display image
dillo (X) -- web browser

Q: wifi/ethernet connection
A: use wicd, wicd-client, wicd-curses, wicd-gtk (replace network-manager)

Q: hook up a Raspberry PI to a computer through an ethernet cable
A: https://www.raspberrypi.org/
RaspPi:
- /boot/cmdline.txt : add "ip=169.254.0.xxx" at the end of the first line (e.g.: xx = 99)
Macbook:
- Network => USB ethernet => connect AND/OR (advanced => renew DHCP lease)

Q: what are the IP around?
A: arp -a

Q: my IP address?
A: hostname -I (upper-case "i")

Q: How to install OpenCV?
A: http://blog.mafrog.info/ (it takes ~10 hours)

Q: Opencv demos do not work...? Seems raspPI camera is not active.
A: to enable the raspberryPI camera: sudo modprobe bcm2835-v4l2

Q: How do I install ROS?
A: http://wiki.ros.org/
on step 2.1, prefer the ROS-COMM method (not the Desktop method)
note (for the record): I initially had a problem with liblz4-dev, (error when launching LZ4 fuzzer), which disappeared when retried (possibly I missed the previous step and this should not occur -- I write it down here for future reference if need be)
Section 3.2 "Adding released packages" also generated some errors, to reproduce the error, type:
cd /home/pi/ros_catkin_ws/build_
Remember to source the newly installed file: source /opt/ros/indigo/setup.bash

Q: How to enable thymio control (for sending command)?
A: asebamedulla "ser:device=/dev/ttyACM0"

Q: How to clone from one SD-card to another (e.g. for cloning an initialized SD card)
A: (Tested on MAC only)
More on: http://computers.tutsplus.com/
1. To build an image from a already initialized SD card:
diskutil list
sudo dd if=/dev/diskXXX of=~/Desktop/raspberrypi.dmg — XXX is SD disk number
Remark: the dd command takes (a lot of) time
2. To Initialize a blank SD card from a disk image:
diskutil list
diskutil unmountDisk /dev/diskXXX
sudo newfs_msdos -F 16 /dev/diskXXX — CAREFUL NOT TO SELECT THE WRONG DISK
sudo dd if=~/Desktop/raspberrypi.dmg of=/dev/diskXXX — very very long
Remark: while cloning, you use ctrl-T in the terminal to know where you are

Q: How can I set the Pi camera controls? (useful as the camera is in auto-mode by default)
A:
sudo modprobe bcm2835-v4l2 (so that /dev/video0 maps the camera)
v4l2-ctl -l
v4l2-ctl --set-ctrl (...)
Remark: check with "raspistill -o image.jpg"
Remark 2: Well, it is actually quite complex to set the camera. Check here for discussion/hints: https://www.raspberrypi.org/

Q: How can I set a fixed IP address for one RPi in a WLAN network?
A:
- Edit the /etc/network/interfaces file so that it looks like this:

auto wlan0

auto lo

iface lo inet loopback
iface eth0 inet dhcp

allow-hotplug wlan0
iface wlan0 inet static
address 192.168.XXX.YYY
netmask 255.255.255.0
gateway 192.168.1.1
wpa-conf /etc/wpa_supplicant/wpa_
iface default inet dhcp

where XXX and YYY are numbers between 0 and 255.

- Edit the /etc/wpa_supplicant/wpa_

ctrl_interface=DIR=/var/run/
update_config=1

```
network={
        ssid="MySSID"
        psk="MyPassword"
        proto=RSN
        key_mgmt=WPA-PSK
        pairwise=CCMP
        auth_alg=OPEN
}
```

where MySSID and MyPassword are your network credentials.

- Reboot the RPi

Q: How can I send messages between RPis through a (W)LAN?
A:
- Set a fixed IP address for each RPi
- Include in your Python code the following functions:

```
import socket, select, struct, pickle

def recvall(conn, count):
        buf = b''
        while count:
                newbuf = conn.recv(count)
                if not newbuf: return None
                buf += newbuf
                count -= len(newbuf)
        return buf

def recvOneMessage(socket):
        lengthbuf = recvall(socket, 4)
        length, = struct.unpack('!I', lengthbuf)
        data = pickle.loads(recvall(socket, length))
        return data

def sendOneMessage(conn, data):
        packed_data = pickle.dumps(data)
        length = len(packed_data)
        conn.sendall(struct.pack('!I', length))
        conn.sendall(packed_data)
```

- To let RPi 192.168.0.2 receive messages include the following statements in your Python code for RPI 192.168.0.2:

```
# Create the server
sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
sock.setsockopt(socket.SOL_
sock.bind(("", 12345))
sock.listen(5)
# Accept one incoming connection and receive one message
conn, (addr, port) = sock.accept()
```

```
data = recvOneMessage(conn)
print data
```

- To send a message from RPi 192.168.0.1 to RPi 192.168.0.2 include the following statements in your Python code for RPi 192.168.0.1:

```
# Create the client
sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
sock.connect(('192.168.0.2', 12345))
message = "Hello, world! From 192.168.0.1"
sendOneMessage(sock, message)
```