

Raspoid: easily combine a Raspberry Pi®, NXT LEGO® and electronics with a Java framework

Dissertation presented by
Julien LOUETTE , Gaël WITTORSKI

to obtain the Master's degree in
Computer Science and Engineering

Supervisor
Pierre SCHAUS

Readers
Chantal PONCIN, Benoît RAUCENT

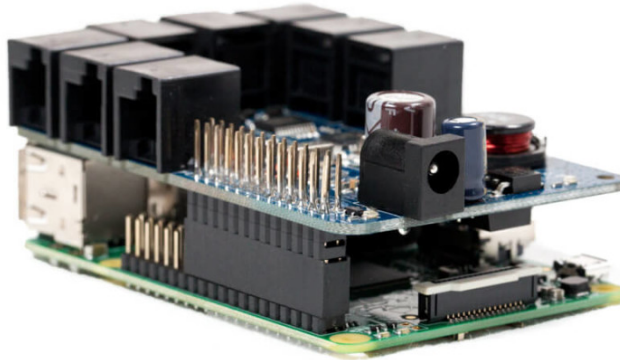
Academic year 2015-2016



ÉCOLE POLYTECHNIQUE DE LOUVAIN

MASTER THESIS
COMPUTER SCIENCE AND ENGINEERING

Raspoid: easily combine a Raspberry Pi®, NXT LEGO® and electronics with a Java framework



Julien LOUETTE
Gaël WITTORSKI

Supervisor: Pierre SCHAUS
Readers: Chantal PONCIN,
Benoît RAUCENT

Academic Year 2015-2016

Abstract

This master thesis explores the possibilities and benefits of substituting a LEGO® MINDSTORMS® brick with a Raspberry Pi® based solution. The core of the thesis consists in the creation of Raspoid, an open-source Java framework, combining MINDSTORMS® components and cheap electronics. The first part presents the context of the project. The second part explains the Raspoid framework operating principle in detail. The third part introduces sensors and actuators integrated in the framework, and their operating principle. The fourth part discusses educational advantages of our low-cost computing platform. A conclusion is that open electronics in conjunction with Raspoid could be a valuable tool for education. A Raspberry Pi® can be used with a BrickPi as an alternative to the LEGO® MINDSTORMS® brick. We show that possibilities offered by the Raspberry Pi® are numerous. This master thesis hopes to offer all students useful tips on creating and developing robots using the Java programming language and proposes a concrete solution to do so.

KEYWORDS: Raspberry Pi, Java Framework, Raspoid, Education, Open-source, MINDSTORMS, LEGO, BrickPi, Electronics, Robotics, Sensor, Actuator.

Acknowledgments

This master thesis is the culmination of our curriculum at UCL. We would like to thank all the people who helped us to achieve this work.

First and foremost, we would like to express our gratitude to Pierre Schaus, our supervisor, for proposing this very interesting master thesis topic and for his guidance along the year. Our grateful thanks are also extended to Chantal Poncin and Benoît Raucent who kindly accepted to be our readers.

Also, we would like to thank Francis Wittorski and Pierre Louette, our fathers, for their careful review and precious suggestions.

We would like to especially thank Fanny and Stéphanie for their daily support. Finally, we would like to thank our family, friends and all the people who supported us throughout the year.

Julien Louette & Gaël Wittorski

June 2016

Contents

Abstract	ii
Acknowledgments	v
Introduction	1
I Context	3
1 LEGO Mindstorms	5
1.1 Presentation	5
1.2 The Mindstorms NXT basic kit	5
1.3 Programming the brick	6
2 Raspberry Pi	7
2.1 Presentation	7
2.2 Raspbian	7
2.3 GPIO header	8
2.4 Connectivity	9
3 BrickPi	11
3.1 Description	11
3.2 Internals	12
4 Objectives & organisation	15
4.1 Objectives	15
4.2 Organisation	15
II Raspoid Framework	17
5 Overview	19
5.1 Specific pin numbering	19
5.2 Dependencies	20
6 BrickPi NXT	25
6.1 BrickPi API internals	25
6.2 UART communications	27
6.3 Message exchanges	28
6.4 Controlling NXT devices	29
7 Additional components	33
7.1 GPIO components	33
7.2 PWM components	34
7.3 I ² C components	36
7.4 Analog components	37

7.5	Camera Pi	38
8	Behavioral programming	39
9	Network utilities	41
9.1	Router	43
9.2	Socket server	43
9.3	Message-like socket server	43
9.4	Pushbullet	45
10	Metrics	47
III	Additional Components - Tutorials	49
11	GPIO - Ultrasound sensor	51
11.1	Operating principle	51
11.2	Example of use with Raspoid	53
12	I²C - Accelerometer & gyroscope	55
12.1	Operating principle	55
12.2	Example of use with Raspoid	58
13	PWM - Servomotor	61
13.1	Operating principle	61
13.2	Example of use with Raspoid	63
14	Analog to digital - Photoresistor	65
14.1	Operating principle	65
14.2	Example of use with Raspoid	66
15	Camera Pi	67
15.1	Raspberry Pi camera module	67
15.2	Example of use with Raspoid	67
IV	Educational Interest	71
16	Raspberry Pi in education	75
17	Robotics in education	79
18	Raspoid at EPL	81
	Conclusion	85
	Future work	87
	Abbreviations	89
	List of figures	92

Whole bibliography	93
Articles only	95
Books only	95
URLs only	96
Datasheets only	96
A Raspoid - Useful links	97
B Compile Pi4J with a specific baud rate	99
C Create an SD card with the Raspoid OS image	101
C.1 Format your SD card	101
C.2 Flash your SD card with the last Raspoid OS image	101
C.3 Complete the installation	102
D Additional components - Tutorials	103
D.1 LED, LED PWM & AutoFlash LED	104
D.2 Button & touch switch	107
D.3 LCD Display - LCM1602	109
D.4 Joystick	110
D.5 Rotary encoder	112
D.6 Thermistor	114
D.7 Barometer - BMP180	116
D.8 IR receiver & IR transmitter	117
D.9 Accelerometer - ADXL345	119
D.10 Sound sensor - LM358	120
D.11 Buzzer (active - passive)	121
D.12 Tracking sensor - TRT5000	123
D.13 IR obstacle avoidance module	124
E Robot - Proof of concept	127
F BrickPi - Hardware schematics	135

Introduction

Educational robotics

Since 1998, the LEGO company released several robotics kits for the public, allowing to assemble and program small robots: the LEGO MINDSTORMS. These kits provide simple enough ways to build robots, such that they are a good choice for educational and experimenting purposes. They are mainly open-source (both software and hardware) and since their launch, the community has grown and some compatible hardware and software third-party tools have been deployed.

More recently, since 2012, the Raspberry Pi Foundation released several Raspberry Pi computers. These computers are single board computers, usually running under Linux, with a size approaching the size of a credit card. Their small size and their low cost makes them, amongst other things, very suitable for domotic systems and robotics applications. Their primary purpose was to be an educational tool: everything is open-source. Today, there is a large community around the world participating in the project by actively creating extensive hardware and software.

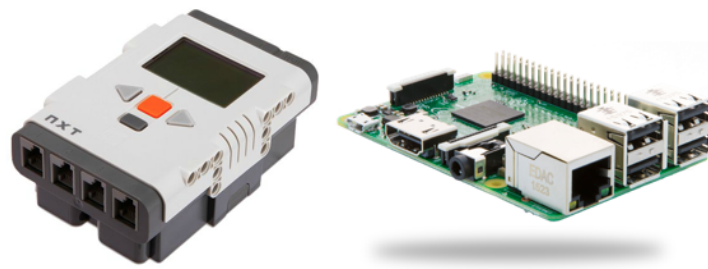


Figure 1: Left side: NXT brick¹; Right side: Raspberry Pi 2².

The best of both worlds

The two systems considered here for educational robotics have different purposes. The Mindstorms kit offers a main brick on which devices can be plugged, and that can be programmed to execute the behavior of the robot. The devices, mainly motors and sensors, are Plug and Play and use a specific connector (RJ12) to be plugged into the main brick. Everything is designed to be very well integrated, and easy to control. A specific programming environment is delivered by LEGO and can be used to build the program with a "visual programming language", as well as to deploy it very simply on the brick. While this approach is simple, the sensors and motors used here need to be specifically developed for the Mindstorms kit: they are more expensive, and the choice is rather limited.

Compared to the Mindstorms, the Raspberry Pi offers much wider flexibility. It can run a regular OS like Linux, and it offers various standard connectivity means such as USB, HDMI, RJ45, GPIO pins, UART, I²C, PWM, etc. It is possible to use all the programming languages, development environments and applications that are available for the selected OS. We can potentially use any electronic devices or circuits that can be interfaced with one of the Raspberry Pi connectivity means. Usually, these additional components are cheaper than Mindstorms components. While the flexibility is much better with the Raspberry Pi than with Mindstorms electronics, it is also a bit less Plug and Play when it comes

¹Source: <http://shop.lego.com/en-CA/NXT-Intelligent-Brick-9841>

²Source: <http://be.farnell.com/fr-BE/buy-raspberry-pi>

to use components such as motors and sensors. The Raspberry Pi has not been designed specifically for robotics, its purpose is much wider. Some components need low level implementation to be used, and it can be complex to achieve the same quality and integration like the Mindstorms motors for instance.

Based on these considerations, we would like to enjoy a tool that provides ease of use, versatility and limited cost. This brought us to develop the Raspoid framework: it is a combination of hardware and software running on top of a Raspberry Pi which allows to connect Mindstorms components as well as regular electronic components. The API provides a simple way to interact with Mindstorms motors and sensors, and also with a set of common and cheap devices widely available. It is build in a modular way, with the aim of being straightforward to add a new components. It comprises utilities to control a robot in a natural way through the behavioral programming. Network communications are made simple thanks to utilities relying on regular TCP sockets, HTTP requests and others such as Pushbullet commands.

raspoid.com

In parallel to browsing this document, the reader is invited to visit the raspoid.com website that we have developed throughout the work. This website is a significant part of the project and it contains all necessary information to enjoy with Raspoid: tutorials, code, libraries, documentation, etc.

What's on the menu

This document describes the result of our work. The educational aspect is essential in this work, therefore it will be emphasized all along the report. We will strive to write it in a way such that newcomers can learn during their reading. Readers already accustomed with Mindstorms kits and Raspberry Pi may skip the first two chapters.

- In the first part, we will set the scene by explaining in more details the Raspberry Pi and Mindstorms components which are relevant to us. We will explain why it is interesting to get rid of the regular brick and replace it with the combination of a Raspberry Pi and a BrickPi (presented in chapter 3).
- The second part is presenting the Raspoid framework itself, its features and how it operates. Each main part of the API will be covered, that is the BrickPi, the additional components, the behavioral paradigm and the network utilities.
- Different types of additional components have been integrated in the framework. We classified these components depending on the connectivity they rely on: GPIO, PWM, I²C, analog and the camera. The third part will present one additional component of each type.
- The fourth part discusses educational opportunities of the Raspberry Pi and the robotics in the education field. It will address the educational value that the Raspoid framework can bring to the learners. Some leads are proposed to use it in a project based learning (PBL), such as taught at EPL.
- The conclusion will synthesize what was built and the future work that could be achieved to further improve and extend the framework.
- The size of this report being limited, appendices are used to present additional information.

Note: LEGO® and MINDSTORMS® are trademarks of the LEGO Group. Raspberry Pi® is a trademark of the Raspberry Pi Foundation. In this report, we will respectively use LEGO, Mindstorms and Raspberry Pi to refer to these brands.

Part I

Context

1.1 Presentation

Three major versions of the Mindstorms kits were released: RCX in 1998, NXT in 2006 and EV3 in 2013 [1]. They all work in the same way: sensors and motors are connected to a brick, and a program is executed to manage the system. The different components in the kits have different characteristics and are not fully compatible between a version and another, although some parts might be compatible to some extent. Our work focuses on the NXT version of the Mindstorms kits, because we had access to this kit, and because the LEGO project in the first year at EPL currently uses this kit's version. Unless specifically mentioned, in the rest of the document we refer to the NXT version when addressing the Mindstorms kit.

1.2 The Mindstorms NXT basic kit

The NXT brick (as shown in figure 1, p.1) is the central part of the kit. It features: [2]

- three ports to connect motors,
- four ports to connect sensors,
- a loudspeaker,
- a screen to display information,
- some buttons,
- USB and Bluetooth interfaces to load the programs into the brick,
- a battery.

The motors and sensors are plugged through an RJ12 plug with a modified poka-yoke. The cable conducts the power and the communication between the brick and the devices. Some devices are analogue whereas other devices use the I²C protocol for control and communication[3].

The servomotors allow to move or control a mechanical part of a robot. They can run forward and backward at a defined speed. They use an internal encoder providing information from which we can deduce the angle or the number of performed rotations. This is useful for instance to travel a specific distance, or to synchronize more than one motor.

The touch sensor is the simplest sensor. It informs whether it is pressed or not.

The light sensor detects the light intensity and provides this indication with a value ranging from 0 to 100 (0 is darkest and 100 is brightest).

The sound sensor measures a sound pressure up to 90 dB and provides a value in dB or dBA. A dBA scale gives a measure of the sound felt by the human ear, while a dB is only a physical measure of the sound wave intensity. The dBA scale is a physical measure corrected with the response of the human ear [4] (cf. Fletcher curve).

The ultrasonic sensor measures the distance between the sensor and an object (up to 255 centimeters). It basically works by sending a sound wave on an object and listening for its echo. A measure of the elapsed time between the wave sound sent and the echo received, allows to deduce the distance.

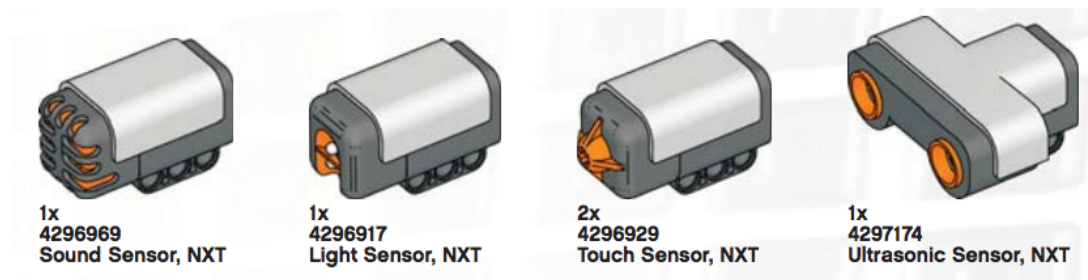


Figure 1.1: NXT sensors, from left to right: sound sensor, light sensor, touch sensor and ultrasonic sensor [2].

Some third party motors, sensors and accessories are also available for purchase. Amongst others, we can find some of them at Mindsensors¹, Dexter Industries², Hitechnic³ and Generation robot⁴.

1.3 Programming the brick

As the Mindstorms kit is an educational project, it comes with an official "visual programming environment" which enables to easily create a program [2]. The control sequences are created by doing a drag and drop of elementary blocks, by configuring them and linking them together. When the program is ready, the user can deploy it from the IDE (distributed by LEGO) by uploading the program to the brick. While this graphical IDE makes it very handy to anyone to start learning and programming, we cannot write plain code with this official IDE.

However, since the brick firmware is a virtual machine interpreting bytecode, numerous other development tools can be used to develop in different programming languages like Python with jaraco.nxt⁵, Ocaml with OCaml-mindstorm⁶, C++ with NXT++⁷, etc. The firmware itself can be replaced with a custom one, therefore allowing to implement custom features or programming patterns. This is the approach that several projects are adopting like ROBOTC⁸ allowing to program in C and leJOS⁹ in Java. The leJOS framework brings some interesting features from the Java programming language itself like threads but also some programming patterns like the behavioral paradigm to control the robots: it influenced us for the realization of the Raspoid framework.

¹<http://www.mindsensors.com/>

²<http://www.dexterindustries.com/>

³<https://www.hitechnic.com/>

⁴<http://www.generationrobots.com>

⁵<https://github.com/jaraco/jaraco.nxt>

⁶<http://ocaml-mindstorm.forge.ocamlcore.org/>

⁷<https://github.com/corywalker/nxt-plus-plus>

⁸<http://www.robotc.net/>

⁹<http://www.lejos.org/>

2.1 Presentation

The Raspberry Pi Foundation is an organization created in 2009 to promote computer science learning through educational tools. The contribution of the foundation in this domain is a series of single board computers that can be purchased at a very affordable price (\$35¹), and a bunch of educational materials like tutorials², documentation, projects, etc.

A specific Linux distribution (Raspbian)³ has been created by some Raspberry Pi's fans, and is officially supported by the foundation.

Since 2012, the year the Raspberry Pi A came out, seven models were released. The latest is the Raspberry Pi 3, presented in February 2016. For this work, we mainly used the Raspberry Pi 2. We will focus on this model but a lot of insights we give for this model are relevant for other models.

The Raspberry Pi 2 was released in 2015 and is the first Raspberry Pi model shipped with an ARM processor. It can thus execute all the OSes compatible with this architecture, that is all the major Linux distributions and even Windows 10. Here are the main specifications of this model⁴:

- A 900 MHz quad-core ARM Cortex-A7 CPU,
- 1GB of RAM,
- 4 USB ports,
- an Ethernet interface (10/100 Mbit/s),
- a CSI connector (image acquisition),
- a Micro SD card slot,
- and 40 GPIO pins combining GPIO, UART, I²C, PWM and SPI.

2.2 Raspbian

As said earlier, there are numerous OSes supported by the Raspberry Pi. However, the most commonly used is the Raspbian distribution. As its name suggests, it is based on the Debian distribution and customized specifically to run on the Raspberry Pi. This is the distribution we used to develop and test the Raspoid framework. Everything described later will be from the prospect of this distribution.

Raspbian is not a real time OS: we cannot set some hard time constraints on processes. Raspbian uses a preemptive scheduler which decides when a program gains control. This can be an issue when a precise timing is needed for a program to work properly. When using Raspbian to run Raspoid, we recommend to run only a minimal set of processes along the program controlling a robot, so that the response time remains acceptable. As discussed in future work, a solution to investigate could be to replace the Raspbian kernel with a real time kernel or use a distribution specifically tailored for this purpose.

¹<https://www.raspberrypi.org/blog/raspberry-pi-2-on-sale/>

²<https://www.raspberrypi.org/resources/>

³<https://www.raspbian.org/>

⁴<https://www.raspberrypi.org/products/raspberry-pi-2-model-b/>

The Raspbian distribution has several features and utilities worth mentioning:

- some firmware and kernel modules allowing to take full advantage of the Raspberry Pi capabilities,
- an easy way to parameter the system via a config.txt file (used when loading the system),
- the ability to install packages available in the Debian distribution,
- a "raspi-config" tool, allowing to perform useful operations such as expanding the space used by the system on the SD card, enabling/disabling the GUI or Scratch, enabling/disabling the Camera support, and loading modules for physical computing for SPI, I²C and UART.

2.3 GPIO header

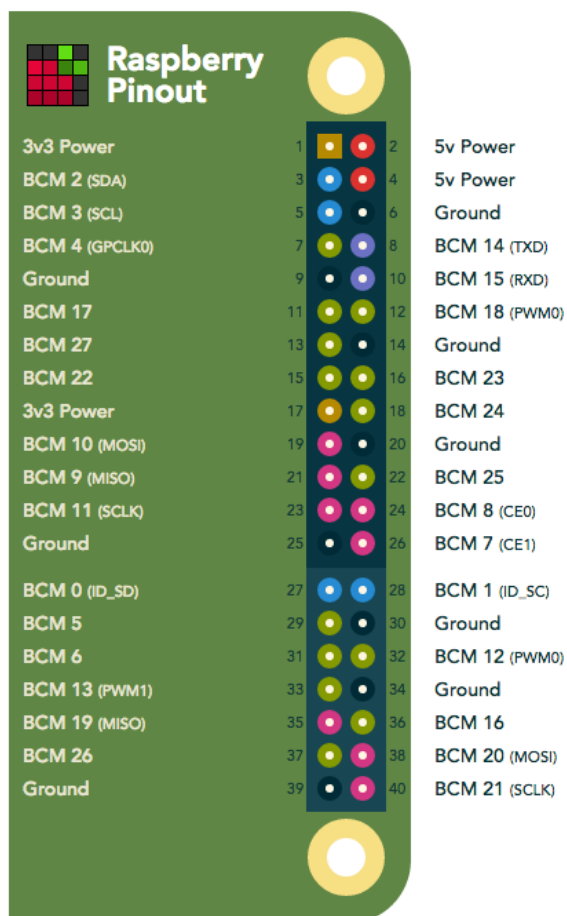


Figure 2.1: Raspberry Pi 2, Pinout[5].

GPIO stands for **General Purpose Input/Output**. These pins are available to connect additional components. They help to assemble and compose circuits without soldering. As their name suggests, they can either be configured in input mode to retrieve information or in output mode to send digital signals.

A GPIO pin can only detect or provide a binary state (low or high voltage). In order to interface an analog component, an ADC (Analog to Digital Converter) is then required.

It is important to mention that in the Raspberry Pi, the low level is 0V and the high level voltage is 3.3V, with no extra tolerance [6]: if this level voltage is exceeded, damages will be caused on the Raspberry Pi.

An important feature of a GPIO pin is the pull up and pull down resistors. When using a GPIO pin in input mode, if there is nothing connected to the pin, the value of the pin can be measured between the low and high voltage, because it has no reference. This behavior is called floating [7]. To avoid this behavior, the Raspberry Pi has some built-in pull-up and pull-down resistors. They can be enabled to force the state of the pin when nothing is connected. For instance, with a push button, this allows to always read the same state value when the circuit is open.

Another important feature is the ability for a GPIO pin in input mode to trigger interrupts. This is useful to immediately react to events without the need to do active polling (and waste resources).

An interrupt can be configured to be triggered when the signal is rising, falling, or both.

The GPIO pin header of the Raspberry Pi is located on the side of the board and contains 40 pins with different capabilities. The figure 2.1 shows the "pinout" layout of this header, with two wiring schemes. The first one is the physical layout, in which all the pins are numbered according to their physical

location. We can see on the figure that the numbers next to the pins are increasing in an ordered way. The second one is the GPIO numbering scheme, in which the pins are named according to the BCM nomenclature.

Here are the different pins types shown in the figure 2.1:

- 28 GPIO pins are available. They are denoted in the figure by "BCM" followed by a number from 0 to 27. Some of these pins superpose some extra functionalities like I²C (BCM 2 and 3), UART (BCM 14 and 15), etc.
- 2 "3.3V Power" pins. They are voltage supplies to power circuits or components.
- 2 "5V Power" pins. They are other voltage supplies with higher voltage, connected directly to the Raspberry Pi's power supply.
- 8 Ground pins, all connected together and acting as neutrals.

The BCM numbering has the advantage of providing a better compatibility between Raspberry Pi versions although the numbering changed between different versions of the model B. Another numbering scheme, coming from WiringPi, handles these changes in a better way. The Raspoid framework offers a way to use any of these three pin numberings (physical, BCM or Wiring Pi).

2.4 Connectivity

While using GPIO to interface electronics circuits allows to build many interesting applications, several advanced interfacing standards are available on the Raspberry Pi such as data transmission in synchronous or asynchronous mode. We will succinctly describe I²C, UART, PWM and CSI. Standard interfaces that are not described here but that could be used with the Raspberry Pi are SPI, JTAG, DPI, GPCLK, SD and PCM.

2.4.1 Serial and UART [8]

With serial communication, a single data bit is sent at a time over a wire. On the Raspberry Pi, a serial bus is available with the pins BCM 14 and 15 (TX to send data and RX to receive). It offers a full duplex bus while minimizing the number of IO lines used. There is no common clock between the two communicating endpoints, therefore it is asynchronous and it requires additional mechanisms to ensure reliability.

An important parameter is the baud rate (symbols per second) which defines the speed at which both ends must transmit data to ensure they can interpret correctly the bits delivered. Each frame is also sent with special bits at the beginning and at the end to delimit the frame. Special bits provide a synchronization mechanism to address the lack of clock. A parity bit can also be used, but it is very limited since it can only detect up to one bit change. The data itself, enclosed in a frame, can vary from 5 to 9 bits depending on the needs, and the endianness can also be configured though it is usually LSB. These mechanisms are adjusted with specific parameters values, so we could say that they are the language of the serial communication. If they don't match, both ends will not be able to understand each other. It is important to remember that, although some reliability mechanisms are built in the serial protocol, it is not fully reliable. Additional mechanisms should be implemented if a truly reliable communication line is required.

On the Raspberry Pi, the universal asynchronous receiver/transmitter (UART) is the component allowing to handle this serial bus. For the TX channel, byte values are written in some registers of the UART, which sends them over the channel as a sequence of bits. Conversely, bits are received sequentially from the RX channel by the UART, which converts them in byte values and store them in registers. The UART interface is a key element for the BrickPi part of the Raspoid framework, as we will explain later.

2.4.2 I²C [9, 10]

Inter-Integrated Circuit (I²C), is a protocol initially designed by Phillips for connecting an electronic circuit or component to a micro-controller. From its initial conception to nowadays, it has emerged as a standard used by many major manufacturers, therefore we can find many components communicating through I²C. An I²C compliant component offers an half-duplex synchronous data bus. This means that it is using a clock to synchronize communicating ends, and the communication is only one way at a time. Like the Raspberry Pi's serial bus, it uses only two wires although it is required that the components share the same ground. The SCL (Serial Clock Line) wire is used to transmit the clock signal, while the SDA (Serial Data Line) wire is used to transmit data bits.

Compared to UART, the advantage is that several slave devices can be connected to one or several masters, while UART allows only two devices to communicate. Each device is uniquely identified by an address. On the Raspberry Pi, it holds on seven bits and allows theoretically 128 addresses [11]. In practice, some addresses are reserved for broadcasting or other purposes. Sur le raspberry la ligne scl correspond au pin BCM 3 tandis que la ligne sda correspond au pin bcm 2. On the Raspberry Pi the SCL line corresponds to the BCM 3 pin, while the SDA line corresponds to the BCM 2 pin. We use them extensively in Raspoid to add custom components, such as accelerometers, barometers, gyroscopes, LCDs, etc. Note that the I²C kernel module is disabled by default on Raspbian, but can be enabled easily by using the raspi-config tool.

2.4.3 PWM

We explained earlier that a GPIO interface has only two possible states: high or low. Pulse Width Modulation is a technique allowing to simulate an analog signal by modulating the duration a digital signal is on (understand high). It can also be used to control electronic components such as servomotors, as presented in chapter 13. The technique relies on a fixed period of time and a duty cycle. The duty cycle is the percentage of time where the signal state is on (high) during this fixed period. By changing the duty cycle, we change the amount of power delivered. The power delivered is not constant but rather intermittent, however the average power delivered is constant and modulated between the low and high values of the digital signal.

A software PWM can be simulated with a GPIO pin by switching it on and off in a very precise timing. Although some libraries implement this feature[12], it is not reliable with a non-real time kernel such as the default one in Raspbian. Fortunately the Raspberry Pi features two hardware controlled PWM pins (BCM 18 and BCM 13). Since only two pins are available, we also used an adapter (PCA9685) with an I²C interface, which is offering 16 PWM channels. However, it comes at the expense of a shorter frequency range available, since it was initially designed to control leds. In Raspoid we use PWM to control servomotors, buzzers, regular and infrared leds.

2.4.4 CSI [13]

CSI was designed for manufacturers in need of a protocol to drive small cameras providing a high bandwidth and a good configurability. This led to the creation of the CSI standard (Camera Serial Interface), for the data transfer between a camera and a micro-processor. The standard describes the one way data transmission protocol, and the two way control protocol which is I²C compliant, both forming the CSI specification. The Raspberry Pi includes a 15 pin header providing the CSI-2 interface on which we can plug the official camera. Theoretically, we could interface any CSI-2 compliant camera, but some new drivers would then be required which is not possible because Broadcom has not revealed the source code needed for this purpose. We used the official camera in Raspoid to take pictures and video streams.

3.1 Description

The BrickPi board is an extension board that can be connected to the Raspberry Pi, and acts as an interface to control the Mindstorms motors and sensors. It was originally founded in 2013 as a Kickstarter project¹ by Dexter Industries², an enterprise manufacturing robotics kits for educational purposes. The hardware and the software are open-source, which makes it hackable. Since its launch, two versions of the board have been released, and they are both compatible with Raspbian, although the first one is not widely available for sale anymore. The figure 3.1 shows the second version of the board. On the left, the board is mounted on a Raspberry Pi 2, and enclosed in a dedicated case. On the right, the raw card is shown from the top.

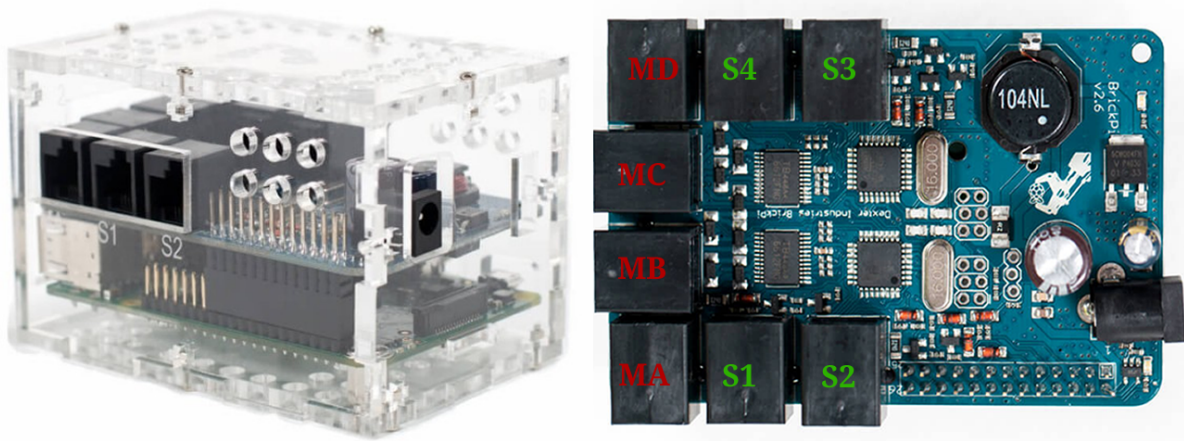


Figure 3.1: Left side: BrickPi+ & Raspberry Pi; Right part: BrickPi+ top view ³

As one can see, the board holds eight connectors, compatible with the specific RJ12 Mindstorms connectors. Four connectors (MA, MB, MC and MD) are dedicated to control the motors, while the four others (S1, S2, S3 and S4) are dedicated to connect the sensors. As shown on the lower right corner of the board, there is a connector using a standard barrel, allowing to connect a 9V or 12V battery to power the motors and sensors. Indeed, an additional power supply is necessary because the Raspberry Pi cannot provide sufficient power, neither manage the current intensity to gear the motor, which can be up to 1A per motor [14]. However, as the 5V power pins of the Raspberry Pi are directly connected to the power supply of the Raspberry Pi, if it is able to deliver a sufficient current, then the BrickPi will be able to only use one of these pins to power itself (as shown in appendix F). Nevertheless, the motors will turn at idle speed and some weird bugs can appear with some sensors. It is also possible to do the opposite and use only the external battery to power both the BrickPi and the Raspberry Pi. It allows to get rid of the transformer connected to the BrickPi, thus releasing the robot from one cable.

¹<https://www.kickstarter.com/projects/john-cole/brickpi-lego-bricks-with-a-raspberry-pi-brain>

²<http://www.dexterindustries.com/>

³Source: <http://www.robotshop.com/eu/fr/kit-brickpi-base.html>

The board fits on the Raspberry Pi by using a 26 pin connector, as shown on the figure 3.1. It is the only mechanical support: it should be handled with care.

With the Raspberry Pi 2, we can see that 14 pins are left accessible. Although the BrickPi uses a 26 pins connector, it only uses a few specific pins as we will see later. The pins that are not used are accessible on the BrickPi board with a pin header stacked on top of the connector. Several BrickPi boards could even be stacked to allow more motors and sensors to be used.

We recommend to use the second version of the BrickPi (the BrickPi+), as it comes with valuable improvements. Here are the main differences between the two versions of the board:

- There is a new stronger set of motor controller chips on the BrickPi+, with more electrical protections [15].
- The dedicated case was initially not well suited to connect the cables, and it was impossible to mount LEGO plastic blocks on it. A new case has been designed to correct this issue.
- The power plug was not initially a standard barrel and we could not track the incoming power.
- In the first version of the BrickPi, a fifth sensor connector was present and referred to as "fast I²C". This connector was directly connected to the Raspberry Pi I²C pins (thus not managed by the BrickPi board) for convenience with some additional third party Mindstorms components.

3.2 Internals

3.2.1 Hardware

The figure 3.2 shows the general architecture of the BrickPi board. The hardware schematics of the BrickPi and BrickPi+ are presented in appendix F. As we can see, the heart of the BrickPi is composed of two Atmega328 micro-controllers [16] similar to the one we find on the Arduino Uno board³. These chips are used because they allow to manage electronic signals in real-time.

Each micro-controller is in charge of controlling two motors and two sensors. The motors operations are controlled by the micro-controller, with the help of a TB6621FNG DC motor controller(not appearing on the figure), by relying on the Pulse Width Modulation technique.

Concerning the sensors, the micro-controller controls them directly by relying either on the I²C protocol, or by using analog signals, depending on the sensor type. For instance, the ultrasonic sensor is controlled with I²C while the touch sensor is an analog sensor. In order to communicate between the Raspberry Pi and the BrickPi, UART, I²C and GPIO pins are used:

- The UART is dedicated to send and receive data asynchronously between the Raspberry Pi and the BrickPi Atmegas. On the BrickPi board, the Rx and Tx wires of the two Atmega328 are soldered together. It can seem strange to have three communicating ends since we explained previously that it was only possible to have two of them with UART. We will explain later how the firmware makes it work. Note that the UART pins of the Raspberry Pi (BCM 14 and BCM 15) are monopolized by the BrickPi board and cannot be used anymore.
- Since its second version, the BrickPi board contains an ADC (an MCP3021 [17]) to estimate the remaining power of the battery. The value returned by this microchip can be retrieved by the Raspberry Pi using the I²C protocol.
- As shown in figure 3.2, the board embeds two LEDs directly controlled from the Raspberry Pi through classical GPIO pins. The LEDs are connected through the pins BCM 27 and BCM 18. The pins can be re-used but the LEDs will blink accordingly to the signal transferred on these pins.

³Arduino UNO Rev3 schematics https://www.arduino.cc/en/uploads/Main/Arduino_Uno_Rev3-schematic.pdf

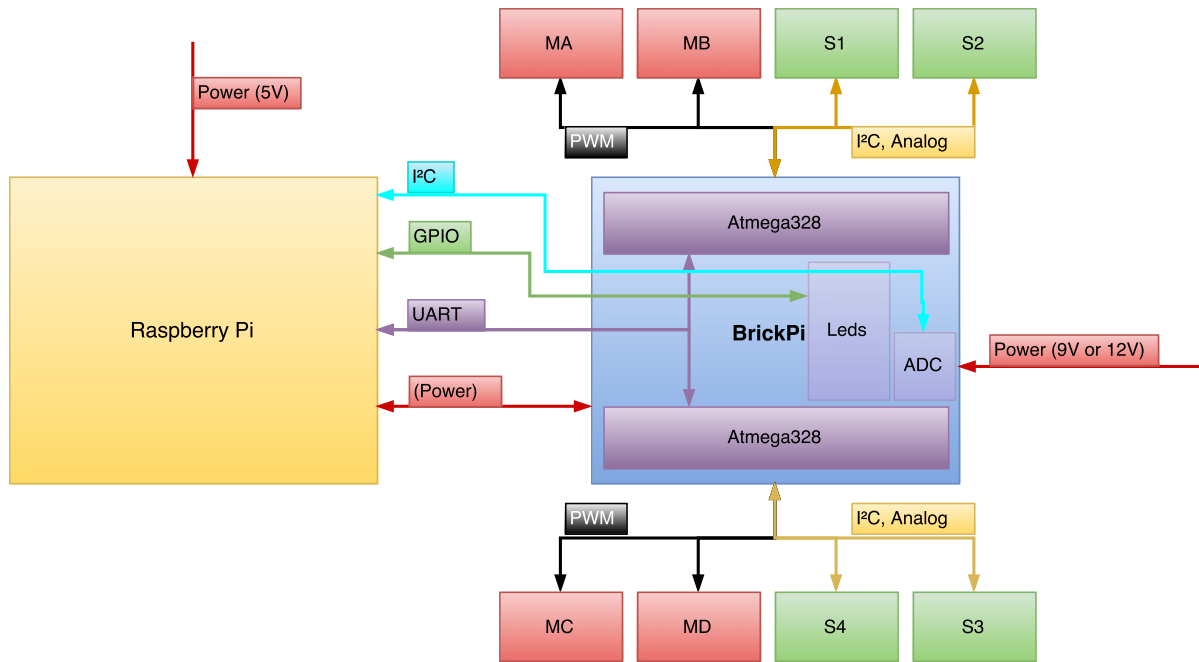


Figure 3.2: BrickPi architecture.

3.2.2 Firmware

The BrickPi firmware⁴ is written in C/C++ and the same code is running on both Atmega328 chips. The firmware is a control loop managing the motors, the sensors and the communications with the Raspberry Pi. It receives some commands that are sent from the Raspberry Pi through the serial interface (UART), performs actions accordingly, and responds with state information.

Since the two Atmegas serial buses are wired together with the Raspberry Pi, each Atmega has a different address configured in its firmware. It will execute and respond to a command, only if the address in the command matches its own address. It always has to send an acknowledgement, with or without payload, to inform the Raspberry Pi that the request was treated and that the serial is now free to be re-used. This way, the Raspberry Pi knows that the response is coming from the chip it previously addressed. The firmware can only respond to commands and never initiate an interaction with the Raspberry Pi, otherwise it could collide with the transmission of another chip.

There is no built-in functionality to identify the type of the devices connected to the BrickPi board. As shown in figure 3.3, a special command is sent once (denoted by 1 on the figure 3.3) at the beginning by the Raspberry Pi to declare the type of each connected device. The firmware then knows which logic it should run to correctly handle the connected devices.

The rest of the time (actually most of the time), it receives an "update values" request regularly from the Raspberry Pi (denoted by ∞ on the figure 3.3), and returns the most recent state of the devices. This update command is also used as a keep alive: the firmware will stop the motors if it does not receive any command from the Raspberry Pi within a configurable interval. Of course, this interval needs to be larger than the maximum inter-arrival command duration, otherwise the motors will stop.

The control logic of the firmware generates signals to control the devices at a constant rate, whereas the Raspberry Pi sends commands with an inter-command delay that can be variable. Once again, this is because the API on the Raspberry Pi is running on a non real time OS. Although we did not alter the firmware, this one is completely hackable and one could add any new functionality.

⁴Source code: https://github.com/DexterInd/BrickPi/tree/master/Firmware_BrickPi/Firmware_2.0

Each micro-controller has a footprint connected to the pins allowing to flash it. Once a new firmware is compiled, the micro-controller can then be flashed by connecting an ISP Firmware Programmer on its footprint (a 6-pin 3x2 header is required).

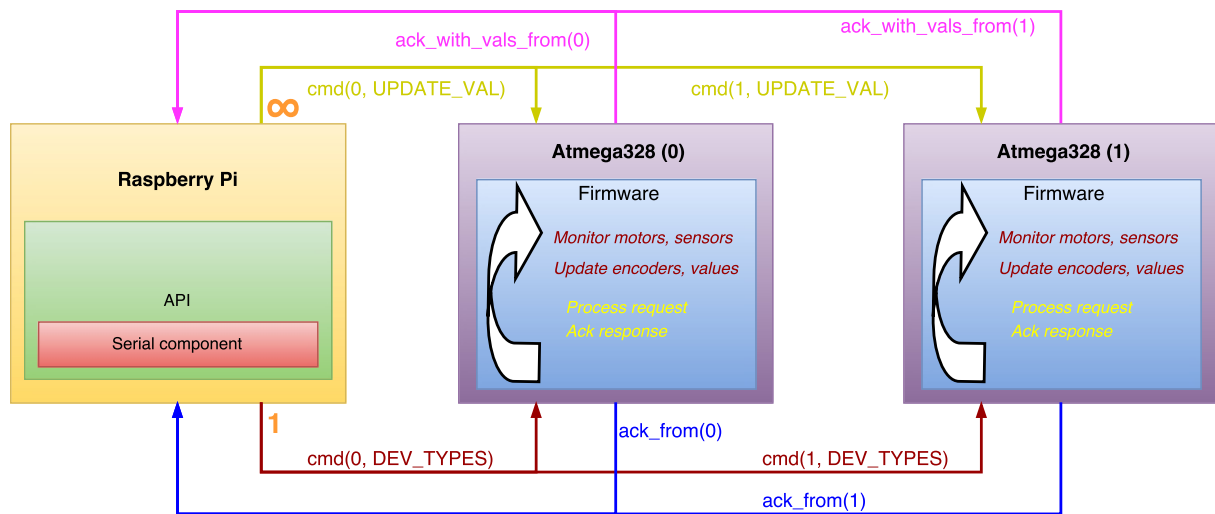


Figure 3.3: Firmware and API interactions. "`cmd(Atmega_dest, Command)`".

3.2.3 APIs

What we call API is the piece of software running on the Raspberry Pi, communicating with the BrickPi and providing high level functionalities. An API was developed by dexter industries for three programming languages: C, Python and Scratch. Other APIs for different programming languages were contributed by third parties: Java, Ruby, NodeJs, Erlang, etc.

As we wanted a language well suited for learning, it was a requirement for us to use Java. Unfortunately, we found that the Java API was very incomplete, that the architecture was not modular, and that it was not leveraging sufficiently the Object Oriented paradigm of Java. That is why in the Raspoid framework we re-implemented the whole API to work with the BrickPi. It allowed us to implement the missing parts of the API and to add functionalities such as listeners, PID control, customizable communication parameters, etc. The implementation is described into details in the next part of this report.

Objectives & organisation

4.1 Objectives

The context being described, we can now state the objectives of our work. We previously mentioned that it would be valuable to combine Mindstorms kits with a Raspberry Pi to benefit from the ease of use, the versatility and the limited costs. The main goal we established is not only to properly control LEGO motors and sensors (NXT) with the help of a BrickPi board, but also that the 1st year students at EPL have a framework to easily build and control a robot. The following sub-objectives have been defined for this purpose:

- Build a simple Java API on top of Pi4J to interact with the motors and sensors through a BrickPi. The firmware of the board may be adapted if needed. A maximum set of native elementary actions should be implemented.
- Build a framework leveraging the behavioral programming technique to provide a higher level interface to program a robot.
- Design and build a scheme to integrate non-LEGO devices in the framework. This would allow to benefit from all the possibilities of the Raspberry Pi in conjunction with the Mindstorms motors and sensors.
- Write some documentation and tutorials to install and explain how to use the framework. The reader should understand how it works under the hood and how to extend it.
- Consider the benefits of using a Raspberry Pi and the Raspoid framework in education.
- Develop a robot as a proof of concept to test and showcase the outcomes.

4.2 Organisation

As learned in the Software Development Project [LSINF2255], Trello was used as a backlog to organize the tasks of research, development and testing. We worked together on the tasks, therefore communication was done directly and very quickly. The GANTT in figure 4.1 summarizes the main tasks and the time we worked on them.

Concerning the toolchain, we used an Eclipse environment, a Git repository hosted on Bitbucket, a Jenkins build server and a SonarQube server. The Eclipse environment was used to develop, compile and push the jar to the Raspberry Pi. Two Git repositories were used for versioning and to synchronize the code and the present document. After a new commit, the new version was automatically built and tested by Jenkins. A lot of useful metrics were also compiled with Jenkins and pushed into SonarQube. SonarQube allows to visualize effectively these metrics, which helped us to assess the quality of the code.

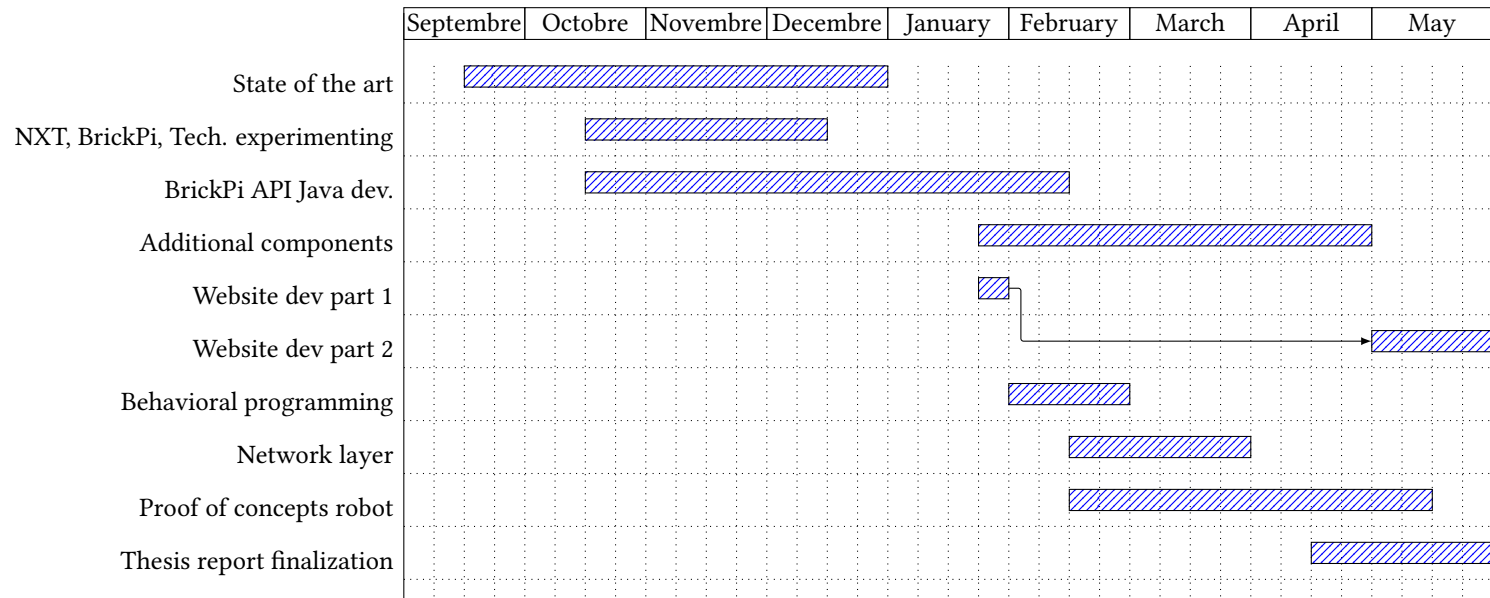


Figure 4.1: GANTT chart.

Part II

Raspoid Framework

Overview

In this second part of the report, we will present the architecture, the internals and how to use the Raspoid framework. During the development phase, we strived to conceive a clear architecture including all the parts of the framework. The objective is also to allow future developers to easily extend the framework.

The six base packages of the framework are represented in figure 5.1. The names given to these packages are self explanatory.

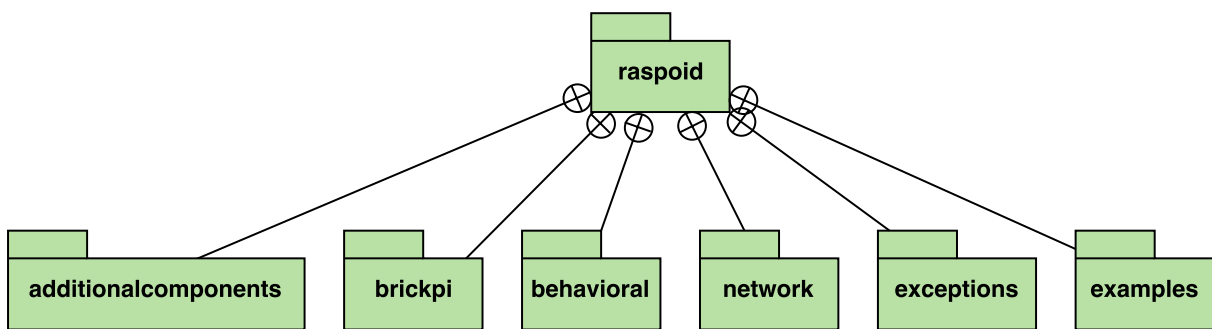


Figure 5.1: Raspoid base packages.

Before presenting each of these packages with useful insights, we give some explanations about the pin numbering abstraction implemented to avoid any confusion with the different pin numberings existing for the GPIO header. We will also present the different external APIs on which our own API relies.

The length of this report being limited, it is impossible to be exhaustive regarding all the methods offered by our API. However, as described in the metrics part (chapter 10) of this report, each method from the API is commented with a complete Javadoc, therefore it should be browsed if more detailed information are needed. This Javadoc is automatically updated after each commit on the production repository and is accessible at the following address: <http://javadoc.raspoid.com/>. We also provide other useful examples/information on the Raspoid.com website.

5.1 Specific pin numbering

As discussed earlier, an important part of the Raspberry Pi is its GPIO header. With recent versions of the Raspberry Pi, one can have access to 40 pins (26 for older versions). Among those pins, eight are connected to the ground, two are used as a 5V power supply and two others as 3.3V power supply. In the 28 remaining pins, some are used for I²C protocol, some others for UART and some for hardware PWM. In order to avoid any sort of problem when using those different pins, we implemented different enumerations to easily access to each kind of pins.

```

1 // One enumeration per type of pin
2 com.raspoid.GPIOPin
3 com.raspoid.I2CPin
4 com.raspoid.PWMPin
5 com.raspoid.UARTPin
6
7 // The PWM pin number 1 is then accessible with the following
8 Pin pwm1 = PWMPin.PWM1;

```

Listing 5.1: Pin numbering schemes.

As explained earlier, it exists various types of pin numbering: the original one, related to the BCM chip used by the Raspberry Pi (BCM numbering); the physical numbering, related to the physical position of the pin on the GPIO header; and the pin numbering used by the WiringPi project, which is widely deployed. It is possible to easily use the numbering of the user's choice to access each pin of the GPIO header. As an example, to access the physical pin number 12, one can use any of the following:

```

1 import com.raspoid.Pin;
2
3 // Those 3 numberings refers to the same pin
4 Pin pin1 = Pin.PHYSICAL_12;
5 Pin pin2 = Pin.BCM_18;
6 Pin pin3 = Pin.WIRING_PI_01;
7
8 // From a Pin instance, one can easily get the correspondence in other numberings
9 pin1.getPhysicalNb();
10 pin1.getBCMNb();
11 pin1.getWiringPiNb();
12
13 // And to get all the informations regarding a pin
14 Tools.log(pin1); // "Name: PWM0 | (physical) 12 | (wiringPi) 1 | (BCM) 18"

```

Listing 5.2: How to access physical pin 12.

5.2 Dependencies

In order to develop our framework, we used different external APIs. While working, we tried existing solutions for each specific aspects of the framework. Sometimes, we decided to develop our own solutions, whereas sometimes it was completely useless to reinvent the wheel when a powerful solution already existed. Here is a presentation of each of the external tools we kept for the project. Some of them such as OpenCV and Pid4j are mostly used for demonstration purposes or very specific use, while some others such as Pi4J and Apache HttpComponents are crucial for some parts of the system. All those frameworks are open-source and licensed without any restriction for the education field. The figure 5.2 graphically shows in which part of the system these external APIs are used.

5.2.1 Pi4J & WiringPi

The Pi4J project allows the Java programmers to interact with the low-level I/O capabilities of the Raspberry Pi. It offers wrapper classes for direct access in Java to the WiringPi library. WiringPi is a GPIO access library written in C for the BCM2835¹, the BCM2836², and the BCM2837³.

¹The Broadcom chip used in the Raspberry Pi Model A, B, B+, Compute Module and Raspberry Pi Zero.

²The Broadcom chip used in the Raspberry Pi 2 Model B. The only significant difference is the removal of the ARM1176JZF-S processor and replacement with a quad-core Cortex-A7 cluster.[18]

³The all new Broadcom chip used in the Raspberry Pi 3, presented in February 2016[19]

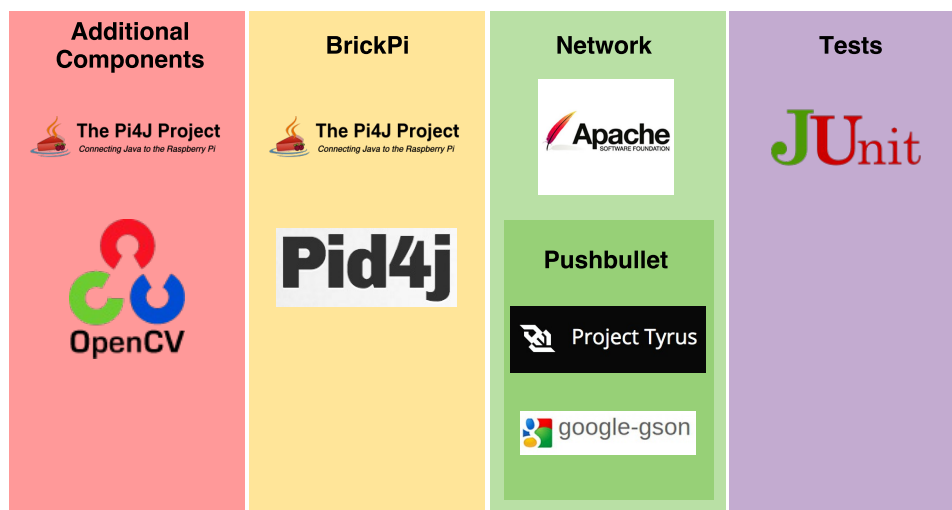


Figure 5.2: Dependencies

Because Pi4J offers the ability to easily control GPIO pins, and can also be used for PWM, I²C and UART communications, it is mainly used for the additional components and BrickPi parts of the Raspoid framework.

To control the UART communications with the BrickPi, we had to compile a new version of Pi4J, with a specific baud rate. This is due to the fact that the firmware on the BrickPi is running at a specific baud rate which is not available in the default release of Pi4J. The procedure to compile Pi4J with a custom baud rate is given in the appendix B, page 99.

Project information

Pi4J

- Project website: <http://www.pi4j.com>.
- License: GNU LGPLv3 <http://www.gnu.org/licenses/lgpl.txt>.

WiringPi

- Project website: <http://wiringpi.com>.
- License: GNU LGPLv3 <http://www.gnu.org/licenses/lgpl.txt>.

5.2.2 Apache HttpComponents

The Apache HttpComponents project proposes a large number of libraries focused on HTTP and associated protocols. We only use two components from this project: the HttpComponents Core and HttpComponents Client. The HttpCore component can be used to build custom client and server side HTTP services, while the HttpClient component is based on the HttpCore and provides components for client-side authentication, HTTP state and HTTP connection management.

We use this tool for the implementation of the HTTP requests in the socket server presented in section 9.3.

Project information

- Project website: <https://hc.apache.org/>.
- License: Apache License 2.0 <http://www.apache.org/licenses/LICENSE-2.0>.

5.2.3 Tyrus Project

A WebSocket is the specification of a protocol allowing bidirectional communications in full-duplex, with only one TCP socket between a client and a server. Initially developed for HTML5, the WebSocket protocol has been normalized by IETF and W3C. Nowadays, all recent browsers implement and support the WebSocket protocol. One of the main advantages of this protocol is its ability for a server to send data to a client in push mode, at the entire initiative of the server.

In the Raspoid framework, we use the WebSocket protocol to interface with the Pushbullet services (presented in section 9.4, page 45 of this report). Tyrus is the Java API allowing us to implement the WebSocket required by Pushbullet.



Project information

- Project website: <https://tyrus.java.net>.
- Licenses: CDDL 1.1 or GPL 2 with CPE (pick which one suites your needs better).

5.2.4 Gson

Gson is an open-source Java library developed to serialize and deserialize Java objects to/from JSON. Initially, Gson was developed for internal purposes by Google. Since 2008, Gson is distributed under the terms of the Apache License 2.0 and is still maintained by Google.

The tool provides simple "toJson()" and "fromJson()" methods to convert Java objects to JSON and conversely.

We use this tool to convert decode/encode received from and sent to the Pushbullet services (presented in section 9.4, page 45 of this report).



Project information

- Project website: <https://github.com/google/gson>.
- License: Apache License 2.0 <http://www.apache.org/licenses/LICENSE-2.0>.

5.2.5 OpenCV

OpenCV (for **Open**-source **C**omputer **V**ision) is a library proposing more than 2500 algorithms for computer vision. It has C++, C, Python and Java official interfaces and is deployed for Windows, GNU/Linux, MacOS X, Android and iOS. OpenCV is mainly developed in C/C++ and takes advantage of multi-core processing. It is licensed under BSD (free for both academic and commercial use) and its large community of developers makes it the leader of computer vision, both for industry and research, all around the world.

In Raspoid, we only use OpenCV for face detection on pictures taken from the Camera Pi. To do this, we compiled a version of OpenCV, fully compatible with the Raspberry Pi. Although we use only a small subset of its capabilities, the users can take advantages of its power and numerous possibilities.



Project information

- Project website: <http://opencv.org>.
- License: BSD (free for both academic and commercial use).

5.2.6 Pid4j

Pid4j is an open-source Java library implementing different PID controllers. A PID (for **P**roportional-**I**ntegral-**D**erivative) controller provides a control loop feedback mechanism to monitor a system over time. It calculates an error value between a measured variable and a desired target. It is used by the Raspoid framework to minimize the errors when moving the NXT motors. The PID controller minimizes the error by adjusting the speed of the motor when the target is nearly reached.



Project information

- Project website: <http://pid4j.org/>.
- License: Apache License 2.0 <http://www.apache.org/licenses/LICENSE-2.0>.

5.2.7 JUnit

JUnit is a unit testing framework for the Java programming language. It allows to easily write repeatable tests, and is integrated by default in well known IDEs such as Eclipse, IntelliJ and NetBeans. It helps to develop test suites that can be run at any time after changes in the code.

JUnit is used to test specific parts of the Raspoid framework.



Project information

- Project website: <http://junit.org/>.
- License: Eclipse Public License 1.0 <http://www.eclipse.org/legal/epl-v10.html>.

6.1 BrickPi API internals

In the `com.raspoid.brickpi.*` packages of the Raspoid framework lies the BrickPi API implementation, which is used for communication with the board. For a regular user of the API, who only uses the API as is, it is pretty simple and exposes access to the board through a main utility class called `BrickPi`; this class represents the physical board itself providing access to the motor ports, the sensor ports and the LEDs. When a motor or a sensor has to be used on a specific port, the user binds an object, representing this device, to the port. Then, interaction with the previously declared physical device through is done through this object. Once everything is bound, initialization of the interactions with the BrickPi is done by calling the `start` method on the `BrickPi` utility class. Conversely, the user tear it down by calling the `stop` method. The following example 6.1 illustrates the basic usage of the API.

```
1 BrickPi.MA = new Motor(); //Binds a motor to the A port of the BrickPi
2 BrickPi.start(); //Initiates the BrickPi
3 BrickPi.MA.setPower(100); // Changes the power of the motor to 100
4 // Do some other interesting stuff
5 BrickPi.MA.stop(); // Done, tear it down
```

Listing 6.1: Basic API usage.

Under the hood, this simple example involves several components working together. The following figure 6.1 sketches the organization and interaction between these components. On the right side of the figure, the two Atmega micro-controllers running the firmware with whom the API will communicate are represented. We already explained that the communication is done through the same UART interface on the Raspberry Pi side, to communicate with both Atmegas. Indeed the Tx (respectively Rx) wires of the chips are soldered together and connected to the Raspberry Pi's Rx (respectively Tx) pin.

Talking directly with the firmware through UART, we can find the `BrickPiSerialTransmitter` component. This component is in charge of sending and receiving bytes over the UART to/from an Atmega micro-controller. It has no knowledge of the data transmitted and it only treats byte chunks. When it receives a byte chunk from the `BrickPiConnector`, it sends it over the UART and waits for the reply. It will always receive some bytes back, because the firmware is always sending an acknowledgment back (with potentially some useful data within) to "release" the UART wires for a subsequent communication. The raw byte chunk received as a reply is then delivered to the `BrickPiConnector` component.

As depicted in the figure 6.1, the `BrickPiConnector` component is acting as an intermediary between the `BrickPi` and the `BrickPiSerialTransmitter`. Its role is to read the state of the different objects which were bound, and to extract configuration values such as the motor power or the sensor type. These values will be enclosed in some messages following a well defined format which is understood by the micro-controller. The message will then be transformed in a byte chunk just before being handed over to the `BrickPiSerialTransmitter` component to handle the transmission. When the transmission is over, the `BrickPiConnector` receives the acknowledgment back, as a byte chunk. At this time, it will do the exact opposite than previously, that is it will process the byte chunk to extract information, and build an acknowledgment message. In case of a pure acknowledgment message, it will just proceed to the next pending operation. However, if the acknowledgment holds some state values, it will update the values of the `BrickPi` component's objects accordingly. In a sense, the `BrickPiConnector` can be viewed as a repetitive synchronizer: at each round it sends the latest configuration values to the micro-controllers

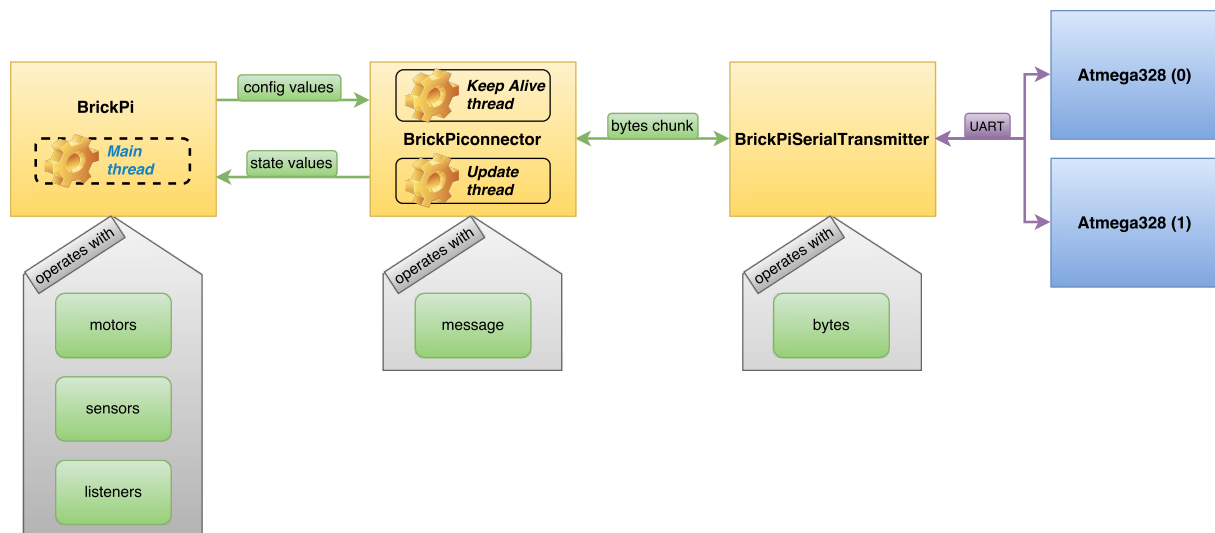


Figure 6.1: BrickPi API architecture.

so that the board acts as instructed by the program, and it updates the BrickPi component's objects with the state values received, so that it reflects the latest state of the BrickPi board. This sequence of operations happens twice during a synchronization round (one for each micro-controller).

We can notice that the BrickPiConnector component represented in figure 6.1 holds two threads: the keep alive thread and the update thread. In order for the BrickPi Java component and for the actual BrickPi board to reflect one another's state, the synchronization rounds should be frequent enough so that their respective state do not mismatch over time with an excessive delay. This is the purpose of the Keep Alive thread, which is in charge of triggering regularly a synchronization round. We call it "Keep Alive" thread because it is at the root of the dynamic behaviour of the API, and also because it is a requirement to regularly send messages to the BrickPi board. Otherwise, if no message is received during a certain window of time, the firmware would perform an emergency motor float. This mechanism is used to avoid running the motors endlessly when no messages are received.

The other thread used in the BrickPiConnector component is the Update thread. Its purpose is to treat the acknowledgments holding some state values, and to update the state of the BrickPi component. One might ask why we are not updating the state directly from inside the Keep Alive thread, after receiving an acknowledgment? The problem is that the update operation could potentially take some time, thus delaying the next synchronization round. Moreover, the time to perform an update operation could vary from one and another, therefore increasing the jitter. We want to keep the polling of the BrickPi board as stable as possible, that is why a dedicated thread is used for the update operation.

At the very left of the figure 6.1 is sketched the BrickPi component, which is the only one to be directly accessed by the user. It is the emerged component of the API, and acts as a control panel. It works like a mixing table: the user plugs the devices to the table, triggers some buttons or commands, and observes some physical and reported feedback on the screen. Internally, it is only used as a holder for the objects manipulated by the API, and for "turning it on/off". Its execution is not performed in a dedicated thread, but in the Main thread of your program, like depicted on the architecture schema. The reason why it is not in a dedicated thread is that most of the time, the actions performed by a program on the API are part of the main control flow of a program; this is why we did not enclosed the BrickPi component in a dedicated thread, but it would be very easy to wrap it inside one if it was ever needed.

After reviewing all the components, one might ask why we did use this specific separation in components, and not another one? The answer is that we tried to gather inside a component the functionalities

operating on the same object's type. The manipulated object's type can be seen as the native "language" of the component. On the architecture schema 6.1, we can see that the BrickPiSerialTransmitter speaks in bytes, that the BrickPiConnector speaks in messages and that the BrickPi handles motors, sensors and listeners. One exception, though, concerns the BrickPiConnector: because it acts as an intermediary between the two other components, it also uses the language of the others, but only to interface with them. This design achieves a better modularity and separation of concerns: suppose we would like a BrickPiConnector which could speak to a distant BrickPi board over the network; the design helps to achieve this by replacing the BrickPiConnector with a component implementing the same functionalities over the network, and with a minimal impact on the other components.

6.2 UART communications

The UART communications are handled by the BrickPiSerialTransmitter class, which is enclosed in the `com.raspoid.brickpi.uart` package. The implementation rely on the Serial abstraction provided by the Pi4J library. This abstraction itself is in fact a wrapper around the WiringPi serial utility. It provides some methods to open a Serial channel and to write/read through it.

Important parameters to mention are the baudrate, the monitoring rate, and the inter-message delay. The baudrate is important because a wrong value would not allow the Serial port to properly encode and decode the signal. The monitoring rate is important to specify how long it should wait before checking if the serial has received new data; this value should be traded off to ensure minimal delay, while avoiding polling too much the buffer. Finally, the minimum delay is the minimum amount of time to wait between sending a byte chunk and reading one from the receive buffer: we could not explain why but this waiting time is required, otherwise the Serial connection hangs (probably because an intrinsic delay is involved when switching between send and receive modes). These constants have been tuned by experimenting to get the best possible results, but they can be tweaked as needed.

We previously explained that the BrickPiSerialTransmitter component had no knowledge of the data transmitted. Indeed, it is completely agnostic in regard of the payload and the packet structure, except for the size of the packet. When sending, it only needs to send the byte chunk as is, but when receiving, there is no way to know the size of the byte chunk to be read from the Serial abstraction. The only way to get it right is to read the length from the data, and to extract the byte chunk with the corresponding size.

Along with the BrickPiSerialTransmitter class, one can find the Packet and PacketFormatter classes. The Packet abstraction embodies the structure that a byte chunk needs to follow, so that the communication with the micro-controllers can happen properly. The packet has a general format, used for all the messages exchanged, and is described in the following figure 6.2.

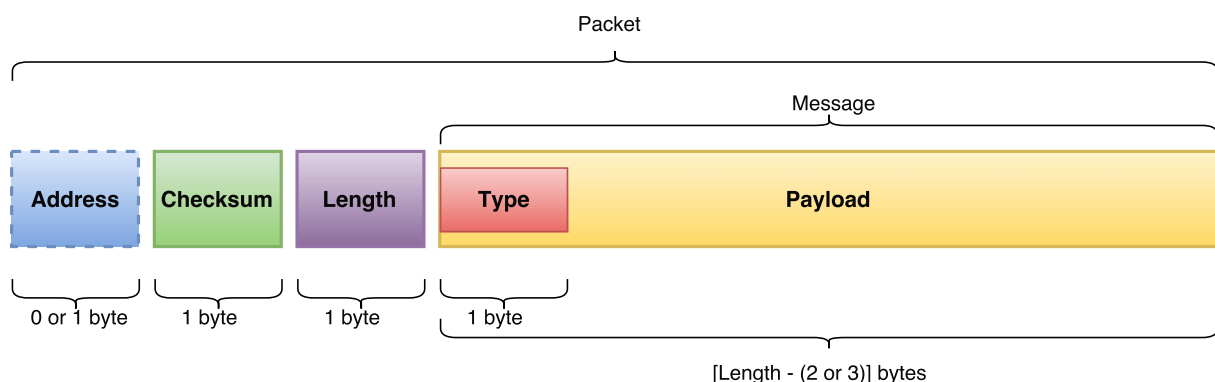


Figure 6.2: Packet format (First byte: only when from the Raspberry Pi to the BrickPi).

As depicted in figure 6.2, the first field is used for the Atmega address. Obviously, this field is present only when sending from the Raspberry Pi to the Atmega controller. The second field is a checksum computed on the packet. We explained earlier that the UART was not completely reliable, therefore it is used as a countermeasure to detect errors. The third field indicates the total length of the packet, so that the receiver knows how many bytes should be extracted. The final field contains the payload, which could be anything, the only recurring pattern being the type byte coming first. In order to provide a simple way for the BrickPiConnector to send and receive messages, the PacketFormatter utility offers two methods to encode/decode messages to/from bytes with the help of the Packet abstraction. It performs the whole encapsulation, decapsulation and error checking operations.

6.3 Message exchanges

We explained earlier the role of the BrickPiConnector component. Before detailing the internals of this component, we need to describe Messages, the objects it manipulates. Each message has a specific purpose in the communication protocol with the BrickPi board. As shown in the packet format 6.2, a message always starts with a byte describing the type of the message. The rest of the message depends on its type, and can have a variable size. The following class diagram in figure 6.3 shows the hierarchy of the messages implemented in the API. Below is a description of each message type. The specific format of each message is not discussed but the implementation is well documented, and should be explored if the reader is eager to get more details.

- The **Message** interface is very simple and defines some common methods to be implemented for each message. Each different message has to provide its type and payload, which will be used by the BrickPiConnector component. The payload will be generated according to the message type.
- The **ValuesMessage** is the message in the API used to regularly poll the BrickPi board to retrieve fresh state values and to keep the BrickPi board alive. It contains the speed to apply to each motor. That is why it receives some suppliers objects providing access to the configured motor speed. When the payload is needed, the current motor speed is encoded on the fly.
- The **EStopMessage** is an emergency stop message which can be sent to immediately float the motors. We don't use it in Raspoid because we noticed that setting the motor speed to zero was more effective and seemed to operate in brake mode instead of float mode. Still it is available and functional for a potential use.
- The **ChangeAddrMessage** is a message type which is not used in Raspoid. It is used to reprogram the micro-controller address. Before re-programming the address, one needs to connect a touch sensor on the first port handled by the micro-controller and maintain it pressed, to ensure that we don't change the address of both micro-controllers. The only use of this message would be when it is wanted to stack multiple BrickPi boards on the same Raspberry Pi. This is not a scenario we have considered, therefore this message remains functional but unused.
- The **SensorTypeMessage** is used to inform the micro-controllers about which sensors are plugged to their ports. Similarly to the ValuesMessage, it receives some Sensors objects in its constructor to be able to generate the payload to send to the Atmega with the right information. It should be noted that the Ultrasonic sensor implementation in the firmware is buggy, therefore a workaround has been implemented in this message. It is commented in detail in the code.
- We previously explained that the Keep Alive thread should send a message within a specific range of time, otherwise the motors would automatically be floated by the firmware. This delay can be configured with the **TimeoutSettingsMessage**.
- Each time a message is sent to the BrickPi, an acknowledgment with the type of the acknowledged

message is sent back to the Raspberry Pi. The **AckMessage** entity represents these acknowledgments, and allows to retrieve the origin.

- When the type of an AckMessage is **AckValuesMessage**, it holds the values specific to sensors as well as the motors encoders values. The **AckValuesMessage** exposes additional methods for to extract these values.

In case a new message would be necessary, the design makes it easy to add it to the API. One should implement the Message interface with a new message type, and it would be ready to be used in the API.

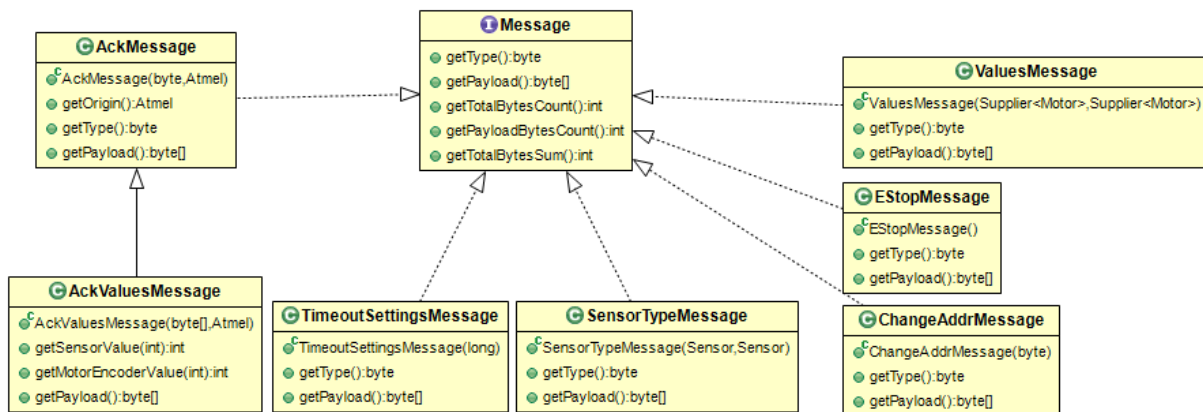


Figure 6.3: Class diagram: messages hierarchy.

Now that the different messages have been covered, we can describe how they are used by the BrickPiConnector component. When it is started, it creates an access to the BrickPiSerialTransmitter component, and initializes the Keep Alive and the Update threads. When the resources are set up, it first sends a TimeoutSettingsMessage to each micro-controller to set a custom timeout delay. Then, it sends a SensorTypeMessage in order to declare the sensor devices connected. The last initialization step consists in launching a repetitive task inside the Keep Alive thread, which will send a ValuesMessage at a fixed rate. Whenever an AckValueMessage is received in response, it is passed to the Update thread to be processed. The Update thread will then use the specific methods of the AckValueMessage to extract the values and to update the BrickPi state. As in the BrickPiSerialTransmitter component, three important constants can be tuned as appropriate for the needs. The first one is the receive timeout constant, that is used by the BrickPiSerialTransmitter to configure the delay to wait for a response: if this delay is outdated, it will consider that the request is lost and will try to send it again. The second one is the motor timeout, specifying the delay after which the motors are automatically floated. This parameter should be greater than the default delay, which is the last parameter: the default delay is used to specify the time to wait between the sending of two successive ValuesMessage. This delay is important for the response time. Once again, it should be traded off between response time, message queue length and polling overhead. The default value has been tuned by experiment, but it may be adjusted for specific requirements.

6.4 Controlling NXT devices

In order to control the NXT devices, the message exchanges handled by the BrickPiConnector could be sufficient. In fact, the official C and Python API, provided by Dexter Industries, are working this way: they use a main loop to send and receive messages, and they retrieve and update some globally available data structures. The approach used in Raspoid is different, and introduces an upper layer providing a set of objects embodying the devices. To use the API, the user virtually repeats what he has

done physically with the board: he binds a device/object on the BrickPi board, and he can subsequently control it directly. There is no need to update a specific data structure in contrast to the official API. The actions performed on the object are echoed on the device while the state of the device is updated into the object, all being done transparently for the API user. This approach allows to abstract all the message exchanges complexity, and provides a more natural way to interact with the devices, most of all in an object oriented way. The different objects/devices are detailed below.

LEDs

The LED object is a bit special because the two LEDs are integrated on the the board. Unlike the other objects, it does not need to be bound, and the two LEDs are immediately available to use through the LED1 and LED2 ports of the BrickPi class. The user can trigger them ON and OFF, toggle them, or send a pulse for a given time. The following code listing 6.2 shows an example of use:

```
1 B<BrickPi.LED1.on();           //Turns on the LED1 on the board
2 BrickPi.LED1.toggle();
3 BrickPi.LED1.pulse(2000);      //Turns on for 2000 milliseconds
```

Listing 6.2: Using a LED.

Motors

An NXT motor is controlled through a Motor object. It needs to be created and bound to one of the four available motor ports (MA, MB, MC, MD) of the BrickPi class. The motor speed is controlled by setting the power with a value ranging from -255 to 255. A positive value turns the motor clockwise whereas a negative value turns the motor counter-clockwise.

The Motor object also provides access to the encoders of the motors. The encoders are internal counters, in the motor, automatically incremented or decremented by the motor while turning. It allows to keep track of the number of revolutions a motor performed and can be used to set the motor at a specific angle. For instance, to perform a complete rotation clockwise, we will ensure that the encoder is incremented by 1440 units before stopping it. Conversely, if we want the motor to perform one quarter rotation counter-clockwise, we will ensure that the motor encoder is decremented by 360 units (1440/4) before stopping the motor.

The encoders are used by the Motor object to perform some actions but can be read by the user. The user can also reset the encoders value if he needs the encoder value to be zero. There are two ways to access the encoders: the first one is to poll the Motor object for this value, and the second one is to register a listener to the Motor. When a listener is registered with the onRotate method, it will be triggered whenever a full revolution has been performed. If more control is needed, then the onChange method can be used to register a listener with a custom defined value range to trigger it. For instance, if we want the listener to be triggered every quarter of a revolution, then we would pass 360 as range parameter along with our listener. The listeners have the advantage not to use resources to check if the encoder changed. They are also used by the Motor object to provide two other utility methods, namely "rotate" and "move". The rotate method can be used to perform a specific number of rotation at a defined speed, while the move method can traverse a chosen distance. The peculiarity here is that both methods use a PID controller to monitor the motor rotation.

A proportional integral and derivative controller (PID) is a mechanism which is iteratively computing an error value (difference between the current state and the set point) with respect to a set point and which will attempt to minimize this error by giving a control output to apply. In the context of the NXT motor, it is used to control the power applied to the motor, so that it stops exactly or close to the target encoder value. Since there is a delay between reading the motor state and stopping the motor, without using a PID, the motor would almost always stop too late. In order for the PID to work properly, three

coefficients need to be tuned. Unfortunately they need to be tuned specifically for each robot, because the applied speed and the weight of the robot has an impact on the motor behavior.

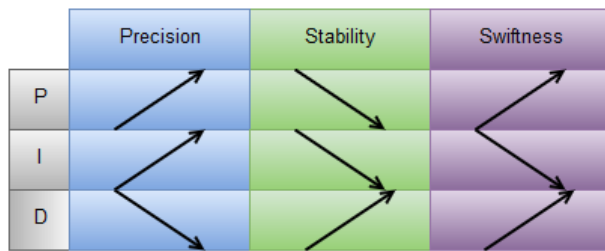


Figure 6.4: PID - Tuning effect when increasing a coefficient.

coefficient, then freeze it, in order to find the I coefficient, which will be frozen as well when searching the right D coefficient.

The following code listing 6.3 summarizes the different features offered by the Motor class:

```

1 BrickPi.MA = new Motor(); //Creates new motor and binds it
2 //Every 500 units, print the new encoder value
3 BrickPi.MA.onChange(500, evt -> Tools.log("Encoder value updated
4 (with a delta >= " + 500 + "): " + evt.getNewValue()));
5 BrickPi.MA.onRotate( evt -> Tools.log("Rotate")); //Prints rotate each lap
6 BrickPi.start(); // Starts the BrickPi
7 Tools.log("Encoders " + BrickPi.MA.getEncoderValue()); //initial encoder value
8 BrickPi.MA.setPower(100); //Sets the power
9 Tools.sleepMilliseconds(10000); // Stays with power = 100 during 10 seconds
10 MA.setDiameter(3); //Diameter of the wheel
11 MA.setPidParams(1.5, 1.2, 1.4); //PID coefficient to use
12 MA.rotate(3, 100); //Rotates three time with initial speed of 100
13 MA.resetEncoderValue(); //Sets encoders to zero
14 MA.move(9.4247, 150); //Moves 9.4247 cm at init speed 150 (= 1 rotation)
15 BrickPi.stop(); //Stops the BrickPi

```

Listing 6.3: Using a motor.

Sensors

Unlike for the motor, several sensors can be used, thus there is a hierarchy among them which is shown in figure 6.5. At the top of the hierarchy is the Sensor abstract class, which is already implementing common operations for sensors. On the left part of the diagram is the branch of RawSensor, defining the analog sensors, while on the right part is the UltrasonicSensor which is an I²C sensor. The light sensor is further decomposed because it can work either with its light ON or OFF. The operation of a sensor remains the same as previously: the user instantiate the corresponding sensor object and binds it on one of the four available sensor ports (S1, S2, S3, S4). Once it is bound, the user can read the value of the sensor in the same way the encoders were read from the motors.

Each sensor has a specific accessor depending on its nature. For instance, the touch sensor has a boolean method `isPressed` while the Ultrasonic sensor has a `getDistance` method. The abstract class `Sensor`, which is the parent of all the sensors also allows to register some listeners to be triggered when the value of the sensor changes. Each sensor also holds a specific byte type value that will be used to declare the sensor to the BrickPi board. The `getType` method is the unique method needed to be overridden

when implementing a new Sensor. All the values and listeners mechanisms are inherited, which makes it easy to add a new Sensor.

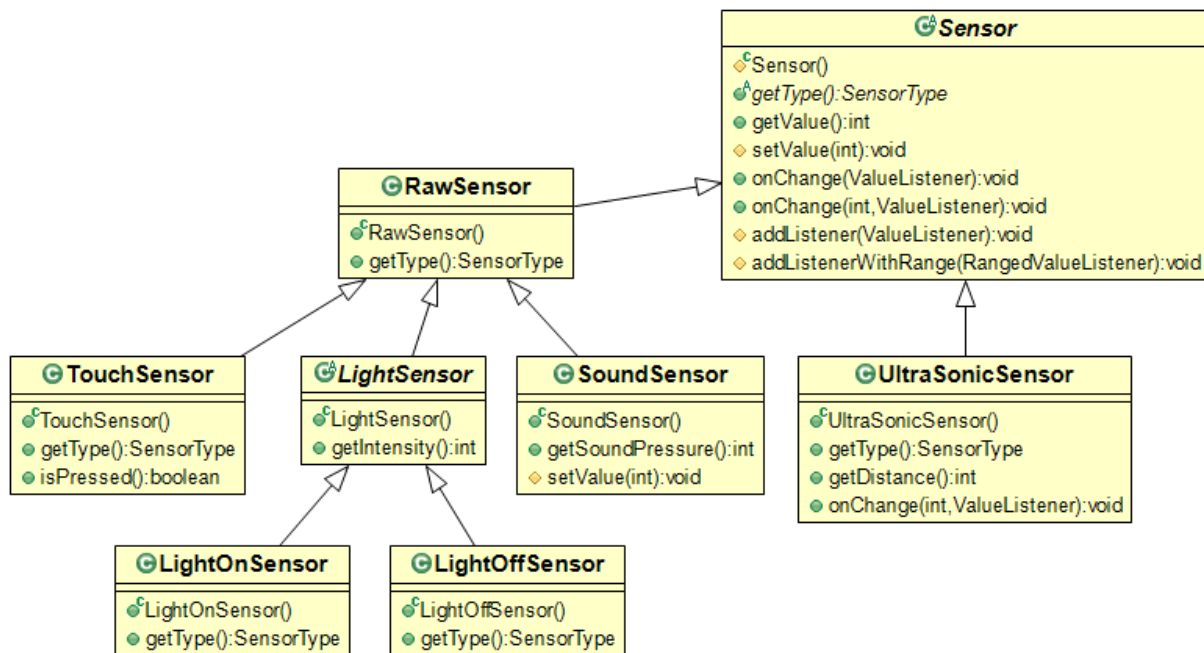


Figure 6.5: Sensors hierarchy.

As the sensors operations are pretty similar to the motor operations, and as there are many sensors, we will not put an exhaustive example here. However, we recommend to take a look at the `com.raspoid.examples.brickpi` package, containing an example for each sensor.

Additional components

A second important part of our framework concerns the implementation of additional components. By "additional components", we refer to sensors, actuators and any kinds of building blocks that may compose a robot and that are not specific to LEGO Mindstorms. As mentioned earlier, there are different kinds of communication interfaces on the Raspberry Pi to control these components. We will present here the architecture of our framework to understand how these different kinds of components can be added to the framework. Our objective is that a user can use the implemented sensors/actuators as they are, or add some new components to the framework by extending or imitating existing ones.

In this chapter, we will present some class diagrams on which we only display useful information (no methods, no class attributes and no secondary classes). On the first class diagram shown below (figure 7.1), we illustrate the organization of the different parent classes composing the architecture of the additional components part of the framework. Each new additional component added to the framework should extend one of these parent classes, according to its type.

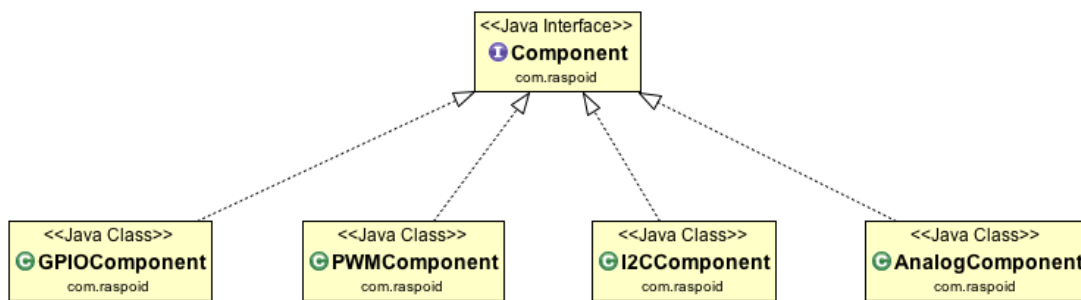


Figure 7.1: Additional components: parent classes.

7.1 GPIO components

We integrated ten GPIO components in the Raspoid framework. The GPIO components are the easiest to control. They only rely on the high/low state of a GPIO pin, and as stated earlier, each GPIO pin can be set in an input or output mode.

When set in output mode, we can control the high/low state of the pin, and thereby control the signal sent to the component. As an example, a LED could shine when the signal is high and stop shining when the signal is low. It works the same way for an active buzzer which will produce noise only when the signal is high.

When set in input mode, it is the component which controls the signal sent to the pin. We can then check the high/low state of the digital signal received on the pin. This check can be performed by using a polling scheme (trivial but inefficient) or by using the interruption mechanism offered by the Raspberry Pi. For instance, when a button is pressed, an interruption is detected on the pin, and the new state can be checked. The same happens when the button is released.

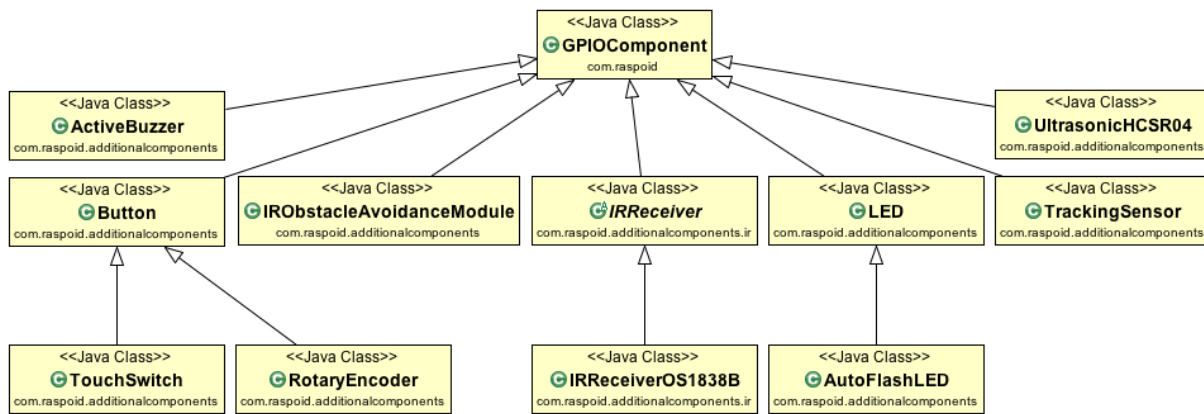


Figure 7.2: GPIO components.

Each component presented in this class diagram (figure 7.2) will be explained in greater details in part III and in appendix D:

- A **touch switch** is a button for which the state is toggled at each touch. First touch: the signal goes high; second touch: the signal goes down; and so on.
- A **rotary encoder** is used here to detect the orientation of a shaft. There are two digital input pins used by this module. The signals on those two pins are analyzed and combined to deduce the direction of new rotations. We use a counter to represent the orientation: when turned to the right, the counter is incremented; when turned to the left, it is decremented.
- The **infrared obstacle avoidance module** is a little chip composed of an infrared emitter and an infrared receiver. It uses the infrared reflection principle to detect obstacles: when there is no object ahead of the emitter, the receiver cannot detect signals; when there is an obstacle, the infrared light is reflected and is then detected by the receiver (it detects obstacles in a 0-7 cm range).
- A **tracking sensor** uses the same principle as the infrared obstacle avoidance module. An infrared signal is emitted, and the reflected signal is received by the detector part of the module. The particularity used here is that black color will not reflect infrared signals. As an example, the sensor can then be oriented towards the ground, and be used to track a black line.
- The **infrared receiver** has been designed to detect infrared signals sent from a media remote. The implementation of an infrared media remote using an infrared-LED is presented in the PWM components part of the framework (section 7.2).
- The **ultrasonic sensor HCSR04** can be used to evaluate proximity with obstacles, using ultrasound propagation and reflection. Chapter 11 presents this useful sensor in detail.

To add a new GPIO component, one only needs to extend the GPIOComponent parent class, or to extend one of the existing components such as the button, the LED or the infrared receiver.

7.2 PWM components

The first use case of PWM (Pulse-Width Modulation) signals is to control the amount of energy sent to a component. As an example, a PWM signal sent to a LED allows to adapt its brightness. The second use case is to send a specific digital signal to a component. A PWM signal can be used in this way to control the orientation of a shaft in a cheap servomotor.

On the following class diagram (figure 7.4), you can observe three "uses" relations (single arrow from PWMComponent to PWMPin, PCA9685 and PCA9685Channel). As discussed earlier, the Raspberry Pi

only features two pins allowing to deal with strict PWM signals. This is a limitation when you want to create a powerful robot using multiple motors, passive buzzers, etc. As a workaround, we decided to integrate a well-known chip to the framework: the PCA9685 [20]. This chip is connected to the Raspberry Pi using the I²C protocol, and is composed of 16 channels allowing to generate 16 hardware controlled PWM signals. With our implementation of the PWM components, each PWM component can be used either with a Raspberry Pi PWM pin, or through a PCA9685 chip.

On the figure 7.3, we show a picture of the Adafruit 16-Channel 12-bit PWM/Servo Driver¹. This module is an easy to use board simply composed of a PCA9685. We bought a generic version of this board for about 9€ on Amazon.fr. It seems to be a very good solution to deal with multiple PWM components from a Raspberry Pi. However, a limitation of the PCA9685 to keep in mind, is its smaller range of available PWM frequencies: the maximum is approximately 1.6 kHz, while it's possible to go up to 19.2MHz with the PWM pins from the Raspberry Pi.

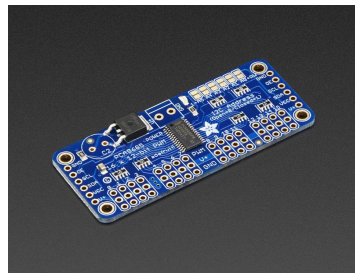


Figure 7.3: Adafruit 16-Channel 12-bit PWM/Servo Driver.

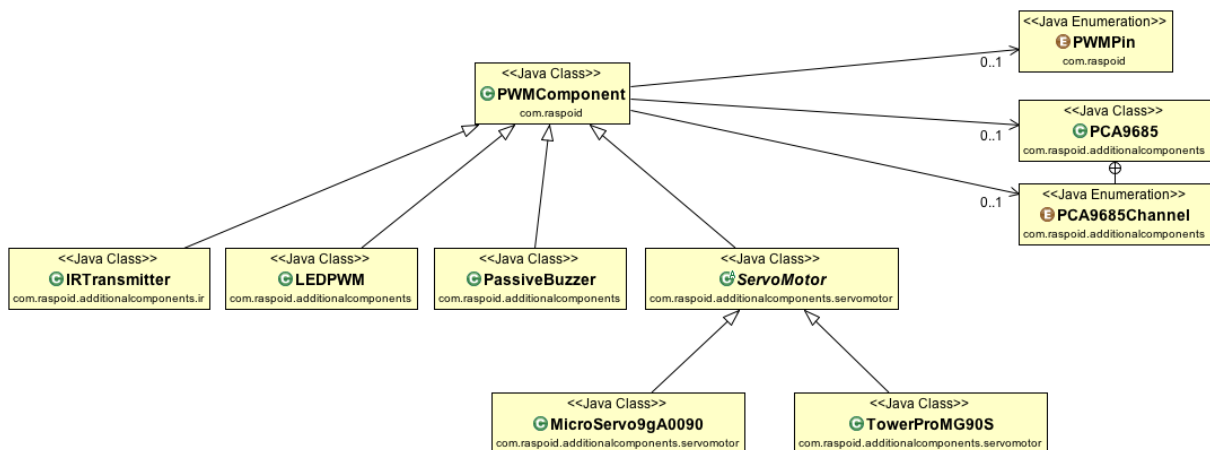


Figure 7.4: PWM components.

Each component presented in this class diagram (figure 7.4) will be explained in greater details in part III and in appendix D:

- An **infrared transmitter** is an infrared LED that should be connected to a Raspberry Pi's PWM pin (the frequency from the PCA9685 is not sufficient for infrared signals). Infrared signals can be sent and detected with the receiver presented in GPIO components section. Our implementation of the infrared transmitter/receiver allows the user to easily control its robot with infrared signals.

¹Website: <https://www.adafruit.com/product/815>

- A **PWM LED** refers to a LED controlled with a PWM signal. It's possible to easily vary the brightness intensity of the LED.
- A **passive buzzer** can be used to control the sound frequency generated by the buzzer. It is possible to integrate a buzzer which plays some specific tones with specific frequencies. Our implementation allows the user to play music by specifying base tones, octaves and notes duration.
- **Servomotors** refers to cheap servomotors controlled by PWM signals. We present these components in detail in chapter 13.

7.3 I²C components

I²C components are more complex to control but often much more powerful. With only two I²C pins on the GPIO header, the seven bits wide address space of I²C theoretically allows 128 I²C addresses. Some addresses being reserved for specific purposes, it is possible to control up to 112 I²C components with a single Raspberry Pi.

Generally, each I²C component requires much more work than the others to be implemented. It is required to find the datasheet for each component, and carefully read and implement the useful features. We implemented six I²C components, for which we tried to have a maximum of functionalities. In the code, each functionality implemented is documented with the corresponding section of the related datasheet. These components can be complex to implement, but they are also more rewarding regarding the functionalities offered.

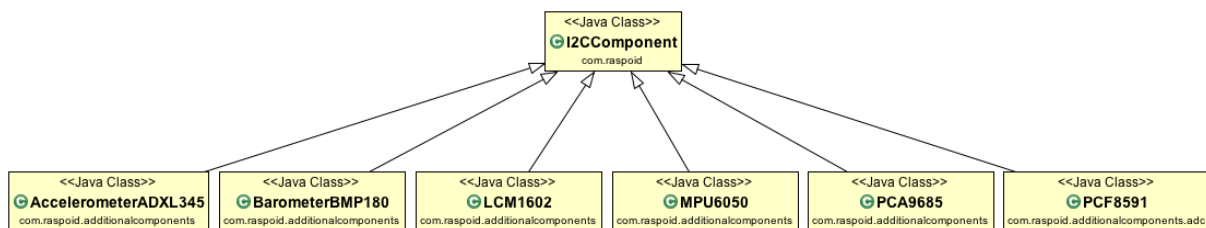


Figure 7.5: I²C components.

Each component presented in this class diagram (figure 7.5) will be explained in greater details in part III and in appendix D:

- The **accelerometer ADXL345** is a cheap accelerometer that can be used to retrieve accelerations for x, y and z axes.
- The **MPU6050** is a powerful sensor combining an accelerometer and a gyroscope. It also integrates a DLPF ("Digital Low-Pass Filter"). This DLPF can make fast calculations directly on the chip, reducing the load for the Raspberry Pi. It allows different levels of accuracy to smooth data. We present this component with much more details in chapter 12.
- The **barometer BMP180** is a digital barometric pressure sensor. It can be used to measure air pressure and temperature.
- The **LCM1602** is an LCD display of 2 lines and 16 columns, which can directly be used with the I²C protocol.
- The **PCA9685** is the chip used to generate up to 16 independent PWM signals, as presented in section 7.2.
- The **PCF8591** is an ADC (Analog to Digital Converter) which can be used to control analog components as discussed in section 7.4.

7.4 Analog components

The Raspberry Pi has no built-in solution to support analog components. By analog components, we refer to components producing an output signal which is not a digital signal. As presented on the class diagram below, to use this kind of components with a Raspberry Pi, it is required to use an ADC. In the Raspoid framework, we chose and implemented the PCF8591. This ADC is composed of 4 analog inputs. It can be used to convert up to 4 analog signals to the corresponding digital values encoded on 8-bits (values are between 0 and 255, depending on the input voltage intensity of the analog signal). These values are stored on the PCF8591 in local registers and can be retrieved by using the I²C protocol. We present the working principle of an ADC in section 14.1. As you can observe on the class diagram (figure 7.6), the ADC has been declared as an interface. One can easily implement a new ADC and add it to the framework in order to it with any AnalogComponent.

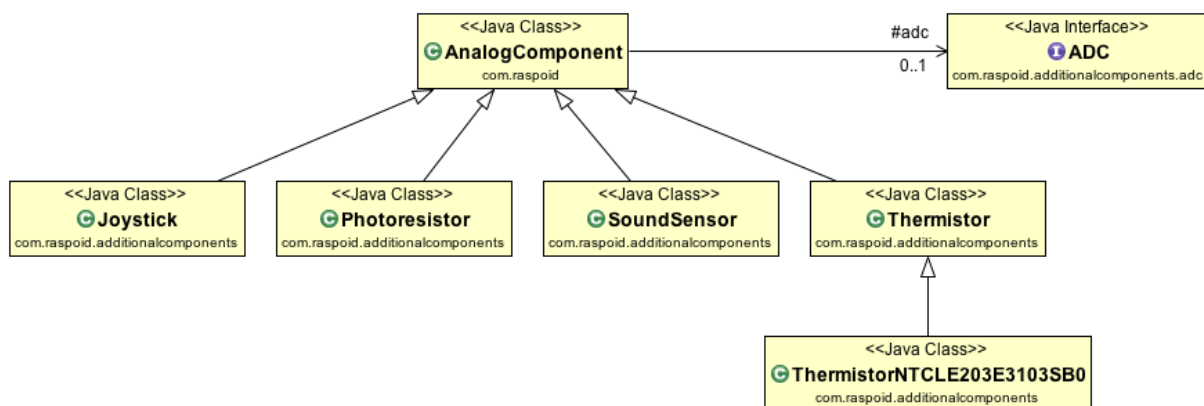


Figure 7.6: Analog components

Each component presented in this class diagram (figure 7.6) will be explained in greater details in part III and in appendix D:

- A **joystick** is mainly composed of two analog outputs. The intensity of the output signal is directly related to the position of the joystick, in the x and y axis (the two analog outputs). From the ADC, we can then retrieve (x,y) coordinates with x and y varying in a range from 0 to 255: (0,0)=bottom left; (126,126)=default position; (255,255)=top right; etc.
- A **photoresistor** is a resistor for which the resistance changes with the intensity of incident light.
- A **sound sensor** is composed of a simple microphone and is able to detect ambient sound intensity by converting audio signals to analog electrical signals.
- A **thermistor** is made of semiconductor materials, for which the resistance varies significantly with ambient temperature. It can be used to detect variations of the ambient temperature.

7.5 Camera Pi

Since 2013, the Raspberry Pi Foundation distributes a Raspberry Pi camera module[21]. This camera board is shipped with a flexible flat cable which plugs itself into the CSI connector of the Raspberry Pi. In the Raspoid framework, we implemented a complete Java wrapper for the "raspistill" and "raspivid" command-line tools provided by the Raspberry Pi Foundation. We also implemented a solution to efficiently stream video contents on the network, and use OpenCV to process images from the camera. The camera module can be used to show previews on an HDMI display directly connected to the Raspberry Pi, to take high-definition videos and still photographs, and to stream video content through the network. OpenCV can also be used, for instance, to detect faces on pictures. The use of the Camera Pi is presented in detail in chapter 15.

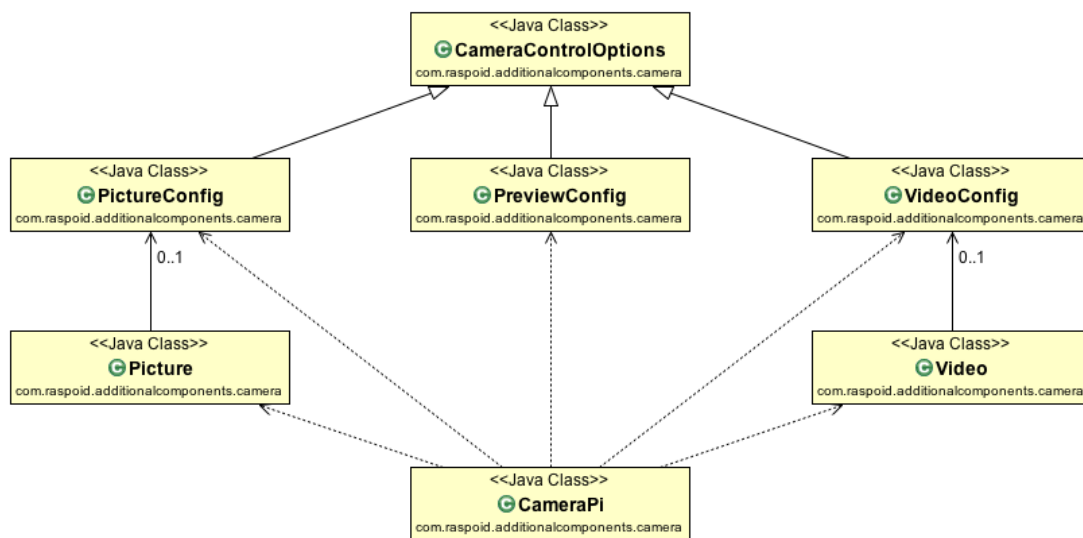


Figure 7.7: Camera Pi

Behavioral programming

Behavioral programming is a paradigm that can be used to program robots in a natural way by specifying behaviors. A behavior implements the operation of a robot when a specific condition is met. With this paradigm, the global behavior of the robot is governed by the combined set of behaviors implemented. All behaviors are independent and can easily be added or removed from the system. This helps to have a very incremental approach when developing the control program of a robot. One can specify at first the most general behaviors of the robot, and then add behaviors handling some more specific situations, thus refining for instance a reaction in response to an edge case.

Several approaches can be used in order to execute the different behaviors. In the approach presented in the paper [22], each behavior is run in a thread, and all the behaviors are executed at the same time. Coordination of the behaviors uses some synchronization points, during which they exchange messages.

The approach which is used in Raspoid is simpler, and inspired from the one used in the leJOS framework[23]. With Raspoid, only one behavior is executed at a time according to its priority. An arbitrator is in charge of continually traversing the list of behaviors, and to execute the claiming behavior with the highest priority.

The following figure 8.1 illustrates the selection process of the arbitrator. The priority is shown on the top right corner of each behavior. On the top left corner is given the state of the behavior: a cross means that the behavior does not claim control of the robot, a down arrow means that the behavior is currently in control, and a green tick means that the behavior claims control. We can see that there are five behaviors registered into the system, ordered by their priority, thus the arbitrator will traverse the list from the right to the left. In this particular configuration, the arbitrator will ask the behavior 3 to yield control, and will pass it to the behavior 4 because it claims control and has a higher priority.

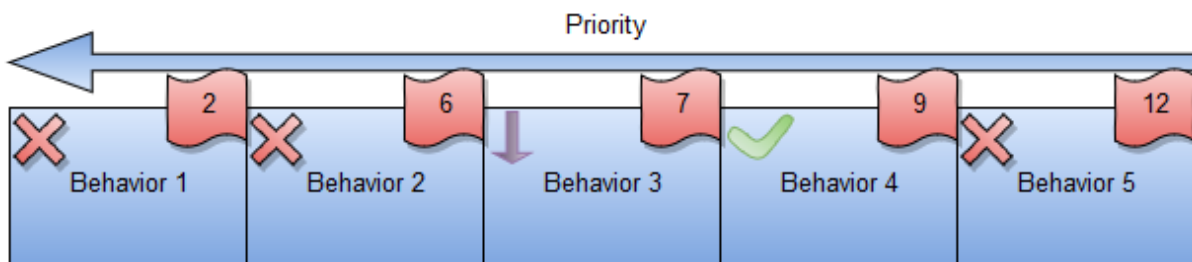


Figure 8.1: Selection of a behavior by the arbitrator

In practice, here is how to use the behavioral programming with Raspoid. The interesting classes are located in the "com.raspoid.behavioral" package. Two interfaces define the methods needed for an arbitrator and for a behavior. Each one of them has an implementation which is called respectively SimpleArbitrator and SimpleBehavior. The SimpleArbitrator can be used as is, whereas the SimpleBehavior needs to be overridden to add some effective behavior. The arbitrator comprises three methods to start, stop and re-launch immediately an arbitration process. The behavior has more methods needing to be overridden, and deserves more explanations:

- **claimControl** is the first method to override when implementing a behavior. It is called by the arbitrator when traversing the list of behaviors to check if a behavior wants to be in control of the robot.
- **gainControl** is the entry point of the behavior when it has been selected by the arbitrator to execute. This is the place where the behavior will perform its logic to control the robot.
- **yieldControl** is called by the arbitrator when it asks the behavior to stop its execution. The arbitrator is polite, therefore it will not kill the current behavior brutally. However when the yieldControl method has been called, the behavior should ensure that the gainControl method stops as fast as possible. Therefore in the logic of the behavior, it should regularly check if the yieldControl method has been called.
- **getPriority** as its name suggests, returns the priority assigned to the behavior. This value will be used to sort the behaviors and break the tie when several behaviors are claiming control.
- **reset** is a utility method used to reset the state of the behavior before it gains control. The behavior can thus do some cleanup from older execution, or simply initialize what is needed before gaining control of the system.

As an example of how the behavioral programming could be used, here is a scenario 8.1 involving two behaviors. The first behavior named MotorBehavior has the lowest priority, and always claims control. Its purpose is to constantly make the motor turn with a specific speed. The second behavior called SoundBehavior has a higher priority but will only claim control when a clap in the hand has been detected. When it gains control it simply executes an anonymous function reversing the direction in which the motor is turning. When the MotorBehavior takes the control back, it will make the motor turn in the reverse direction. The complete example can be find in the "com.raspoid.examples.behavioral" package.

```

1 public static void main(String[] args) {
2     SimpleArbitrator sa = new SimpleArbitrator();
3     MotorBehavior motorBehavior = new MotorBehavior();
4     sa.addBehavior(motorBehavior);
5     sa.addBehavior(new SoundBehavior(() ->
6         motorBehavior.setPower(-motorBehavior.getPower())
7     ));
8     BrickPi.start();
9     sa.start();
10    Tools.sleepMilliseconds(30000);
11    sa.stop();
12    BrickPi.stop();
13 }

```

Listing 8.1: Hand clap example

Network utilities

Communications is an important aspect in robotics. It should be possible to control a robot through the network or to receive information from it, in a synchronous or asynchronous manner. Our architecture for the networking part of the framework is based on the use of a router. The aim of this router is to bring a collection of routes, and to map those routes to some specific logic. The router is at the core of the Raspoid networking functionalities and can be used with any kind of server. A server is like an additional layer implemented on top of the router. We implemented a simple socket server allowing raw or HTTP requests, a message-like socket server, and a Pushbullet server to be able to use the Pushbullet services.

In figure 9.1, one can observe an abstraction of the network working principle. The red part of the schema is specific for each kind of server. The server is the interface with the user (the client). When a client sends a request to the server, the server forwards this request to the router. The router then applies the bound logic, and a plain text response is sent back to the client. Each router has a default "hello" route, for testing purpose, which will respond with a "Hello world !" message: it is then always possible to test any configuration.

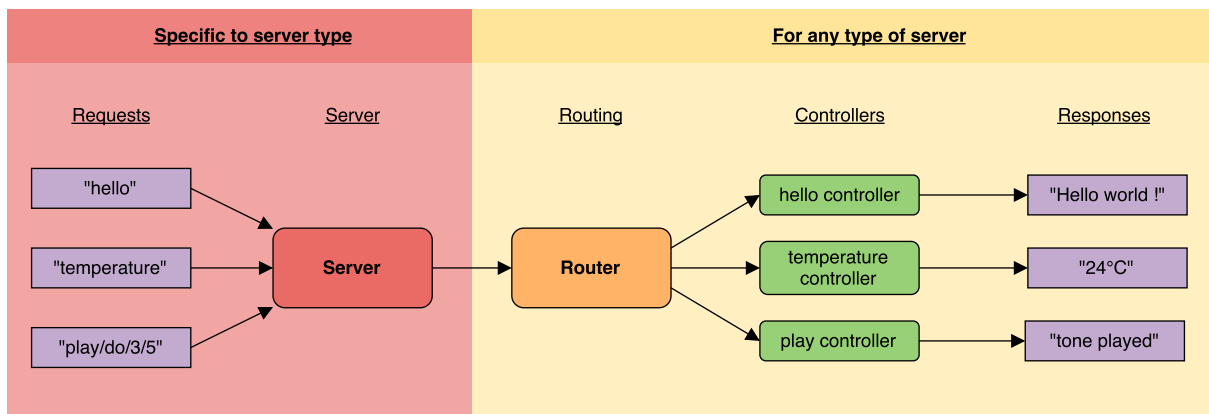


Figure 9.1: Networking - Working principle.

On the first class diagram in figure 9.2, one can visualize the main classes and their organization. In the rest of this chapter, we will present the router and the implemented servers. It is important to state that everything can be extended. It is always possible to add some new server type, to extend existing ones or even to extend the router to add new functionalities.

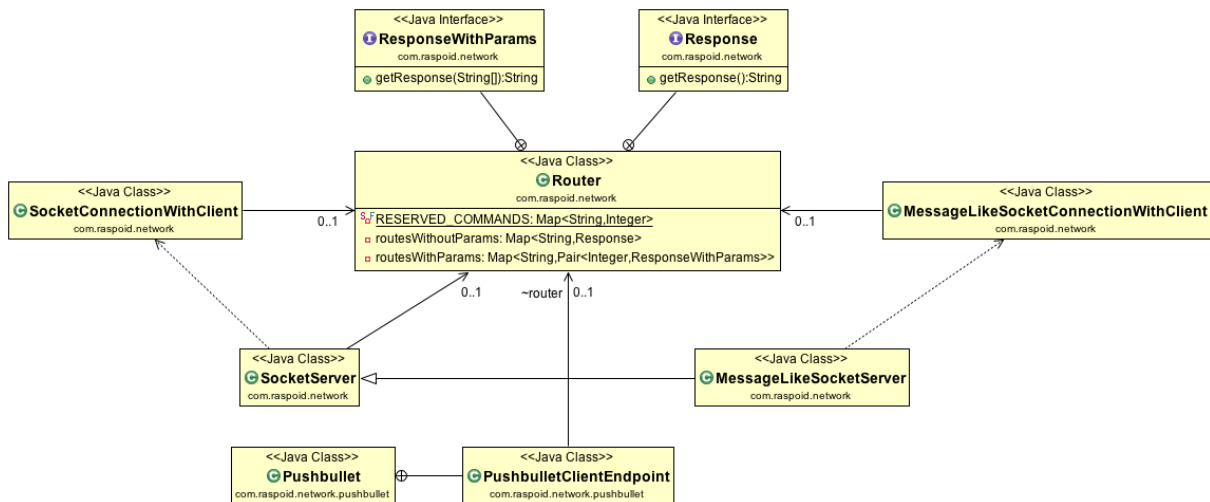


Figure 9.2: Network - Main classes.

The use of the networking part of the framework is as pretty simple, as shown in the following code listing 9.1. We will refer to this code in the next sections.

```

1 public static void main(String[] args) {
2     // A simple router...
3     Router router = new Router();
4
5     // 2 kinds of routes:
6     // without parameters
7     Thermistor thermistor = new ThermistorNTCLE203E3103SB0(new PCF8591(),
8     PCF8591InputChannel.CHANNEL_0);
9     router.addRoute("temperature", () -> thermistor.getTemperature() + "°C");
10
11     // with parameters
12     PassiveBuzzer buzzer = new PassiveBuzzer(PWMPin.PWM0);
13     router.addRouteWithParams("play", 2, inputArgs -> {
14         buzzer.playTone(Double.valueOf(inputArgs[0]), Integer.valueOf(inputArgs[1]))
15     });
16     return "Tone played.";
17
18     // 3 kinds of servers to send requests to and to receive responses from
19     SocketServer socketServer = new SocketServer(5, 80, router);
20     socketServer.start();
21     MessageLikeSocketServer messageLikeSocketServer = new MessageLikeSocketServer(
22     router);
23     messageLikeSocketServer.start();
24     Pushbullet pushbullet = new Pushbullet("YOUR_PUSHBULLET_ACCESS_TOKEN", "Raspoid
    - Example", router);
25 }
  
```

Listing 9.1: Networking with Raspoid - Example.

9.1 Router

A router is primarily a collection of routes together with their corresponding logic executed when the route is called. Two kinds of routes can be added to a router: routes with or without parameters.

On the class diagram presented in figure 9.2, one can notice the `Response` and `ResponseWithParams` interfaces. Those interfaces are implemented as Java 8 functional interfaces, i.e. interfaces containing only one method to allow the use of lambda expressions. The use of lambda expressions allows the user to add new routes with concise, readable and intuitive syntax as presented in code listing 9.1 at lines 8 and 12. In the first case (route without parameter - line 8), the first argument of the `addRoute()` method is the name of the route and the second argument is the logic to apply when the route is called: in this example, the temperature is returned. In the second case (route with parameters - line 12), the first argument is the name of the route, the second is the number of input arguments needed to apply the logic, and the third one is the logic to apply when the route is called: here, a tone with a specific frequency is played for a specific duration.

In the rest of code listing 9.1 (lines 18-22), one can notice the same router instance is given as input argument to the constructor of each kind of servers. As an example, the temperature can then be retrieved with a request like `"http://<server_ip>/temperature"` from a web browser, or with a `"temperature"` message sent from the Pushbullet application to the `"Raspoid - Example"` device.

9.2 Socket server

A socket is used by a node (a client or a server) to control incoming and outgoing flows of data on the network. Each socket is bound to a specific `<IP address, port number>` pair. In our implementation, we use these sockets with the TCP protocol. We implemented our socket server to allow the user to apply simple raw requests or more complex HTTP/1.1 GET requests. When HTTP is not required, a simple socket client can be implemented by the user to send requests to the server. In that case, the packets sent to the server must only contain the signature of the route corresponding to the request to apply on the server.

When HTTP is used, it is automatically handled by the server, by analyzing the first bytes of the received data for each request. Indeed, each HTTP/1.1 GET request starts with a `"GET"` string. The server then replies with a response starting with an `"HTTP/1.1"` header¹. We decided to handle HTTP/1.1 GET requests to allow a user to simply use its web browser to send requests to its robot.

As presented on the schema in figure 9.3, a new thread is created for each new client connected to the socket server.

9.3 Message-like socket server

The message-like socket server extends the previously presented socket server. The need for a message-like socket server is specific to some use cases. We experienced some problems when using a simple socket server, specially when sending data at a high rate. After digging into details, we could observe that some sets of data were merged together when read on the receiver side. The explanation appeared obvious: the problem was to detect the bounds in data exchanged between TCP nodes. The need of a message-like socket server is related to TCP streaming nature.

When two nodes communicate by sending TCP segments, there is no guarantee that when sending a 100 bytes chunk of data, it will be delivered as a unique 100 bytes chunk on the receiver side. There is no way for the receiver to detect the bounds of the data to decode. A bytes chunk sent could be received in

¹`"HTTP/1.1 200\nContent-type:text/plain;charset=utf-8\n\n"` in case of success; `"HTTP/1.1 404\nContent-type:text/plain;charset=utf-8\n\n"` in case of failure.

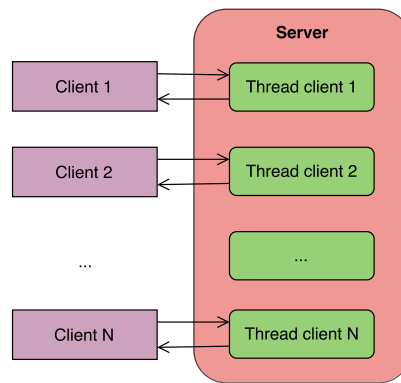


Figure 9.3: Threaded socket server.

several TCP segments, or several data chunks could be merged in a single TCP segment. It is therefore required to deal with these data bounds to ensure reading as many bytes on the receiver side than the number of bytes in the chunk sent. This is done with the implementation of a small protocol on top of TCP.

Our protocol is pretty simple. Each message has a specific format as shown in figure 9.4. The first four bytes are used to define the size of the payload (in bytes). The payload size is defined as a Java primitive int value². The payload must be encoded using the UTF-8 charset. When a receiver detects a new packet, it knows the length of a message and can be sure to read the required number of bytes before delivering them.

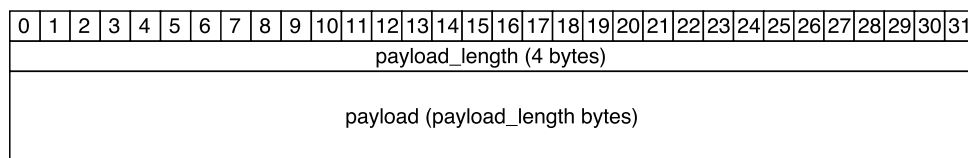


Figure 9.4: Message-like socket server - Message format.

When using our message-like socket server, it is required to implement the client side to deal with our protocol. As an example, in the following code listing, we implemented a simple method sending a new message from the remote joystick, to one of our robots. The robot has a camera support which can be rotated around x and y axes.

```

1 private void updateCameraSupportOrientation(int x, int y) {
2     String request = "update_camera_support_orientation/" + x + "/" + y;
3     try {
4         byte[] message = request.getBytes("UTF-8");
5         outputStream.writeInt(message.length);
6         outputStream.write(message);
7         outputStream.flush();
8     } catch (IOException e) {
9         throw new RaspoidException("Problem when sending new Joystick update: (
10        request) " + request, e);
11    }
12 }

```

Listing 9.2: Sending a message to a message-like socket server.

²32-bit signed two's complement integer, which has a minimum value of -2^{31} and a maximum value of $2^{31} - 1$. Negative values do not make sense and are not used.

9.4 Pushbullet

Pushbullet was created in 2013 in San Francisco by Andre von Houck, Ryan Oldenburg and Chris Hesse³. The aim of Pushbullet is to allow a user to efficiently share information, files and links between his devices: computers, smartphones, tablets, etc. It also allows to share information and messages with other users.

"We believe everything, not just smartphones and computers, should be able to exchange information in real time." - Pushbullet Team.

This universality is made possible thanks to multiple applications and extensions developed by the Pushbullet team to support the majority of existing devices: an application for Android and iOS devices, some browser extensions for Google Chrome, Firefox, Opera and Safari, a native application for Windows, etc. Another strength of Pushbullet is a public API available for developers to use Pushbullet's infrastructure.

With our implementation, it is possible to use Pushbullet services to:

- create a new Pushbullet device corresponding to a robot, and add this device to the list of devices of a user,
- use this Pushbullet device to send messages/requests to a Raspoid robot from any other Pushbullet device (smartphone, tablet, browser extension, etc.),
- receive responses to messages sent to a Raspoid robot,
- receive notifications sent by a Raspoid robot, on all other devices of a user.

To use the official Pushbullet API, an access token is required. This access token grants full access to the user's account. It is important that this token remains secret. It can easily be created from the Pushbullet user's dashboard⁴.

To deal with Pushbullet services, it is required to implement a WebSocket client. As presented in chapter 5, the main advantage of a WebSocket is the ability for a server to send data to a client in push mode. In our case, the server is managed by Pushbullet, and the client is the Raspoid robot. The WebSocket is used by Pushbullet to inform the robot when an event occurs on the user account bound to the robot (a new message received, a new notification sent, etc.). These events do not contain any payload. That is why it is required to listen to each event triggered on this WebSocket and process relevant events related to the managed Raspoid robot.

As presented in the class diagram in figure 9.5, we implemented Push and Device entities. These entities are mappings of Pushbullet Push and Device entities as presented in their public API^{5,6}. When events related to the robot are detected, some HTTP REST requests are sent to retrieve the last received messages. JSON responses from Pushbullet services are then parsed using Gson.

Send a request to a Raspoid robot

As explained earlier, with the Raspoid framework, requests are managed through a router. To communicate with the Pushbullet services, three arguments are mandatory:

- an access token,
- a name for the device corresponding to the robot (the device will be created if no Raspoid device exists with this name),
- a router instance.

³<https://www.pushbullet.com/about>

⁴Pushbullet.com > Settings (<https://www.pushbullet.com/#settings>) > Access Tokens > Create Access Token

⁵<https://docs.pushbullet.com/#push>

⁶<https://docs.pushbullet.com/#device>

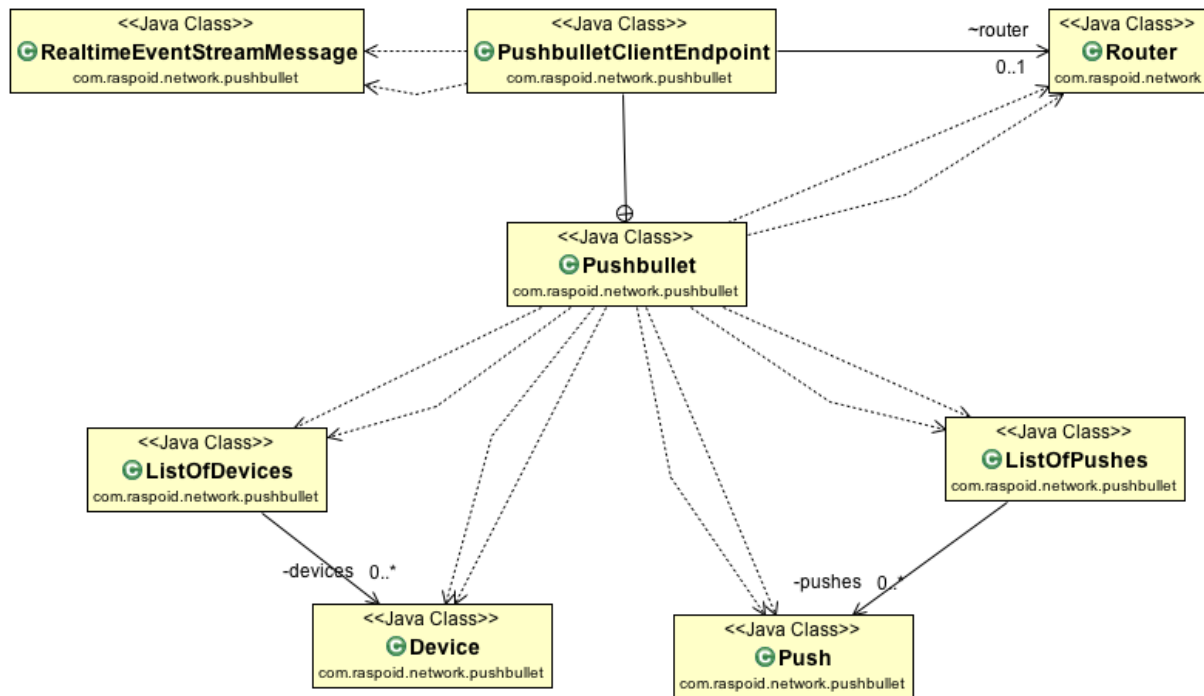


Figure 9.5: Pushbullet - Main classes.

To control a Raspoid robot from a smartphone, it is only required to install the Pushbullet application and to connect with the same user account as the one bound to the access token. The robot will then appear in the list of devices of the user, and it will be possible to send commands to the robot, by simply chatting with it.

Send a notification from a robot

Pushbullet allows a device to send notifications to all the other devices of the user. Our implementation allows the developer to easily send a notification from a robot. In the following example, a notification is sent to all the Pushbullet devices of the user related to the access token, with "Hello !" as title, and "Hello world !" as content. It could be used, for instance, to inform a user about a limit temperature detected by the robot or about any other interesting event.

```

1 public static void main(String [] args){
2     Pushbullet pushbullet = new Pushbullet("YOUR_PUSHBULLET_ACCESS_TOKEN", "Raspoid
  - Example", new Router());
3     pushbullet.sendNewPush("Hello !", "Hello world !");
4 }

```

Listing 9.3: Pushbullet - Send a notification from a robot.

Metrics

The complete code of the Raspoid project is available on a public Git repository hosted on Github¹. We also rent a VPS (virtual private server) at OVH. It is used to host the raspoid.com website, as well as a Jenkins automation server to automatically build, test and deploy a new jar at each commit on the main branch of the Raspoid repository.

We also used the VPS to host a SonarQube installation. This tool is used to analyze code quality. By default, it covers 7 axes of code quality: architecture & design, duplications, unit tests, complexity, potential bugs, coding rules and comments. We extended SonarQube to also apply FindBugs² rules when analyzing code quality. FindBugs is a great tool that uses static analysis to look for bugs in Java code and provide a lot of useful feedback to Java developers.

Among other, SonarQube allows to retrieve advanced metrics regarding the code. We show an overview of those metrics in this chapter. It is possible to check those metrics online at the following link: <http://raspoid.com:9000/>. In figure 10.1, we present a treemap from SonarQube results (for v1). The size of each block is a function of the number of lines of code composing the related package, while the color refers to the density of commented public API. This figure illustrates the care given to document our work with a complete Javadoc.

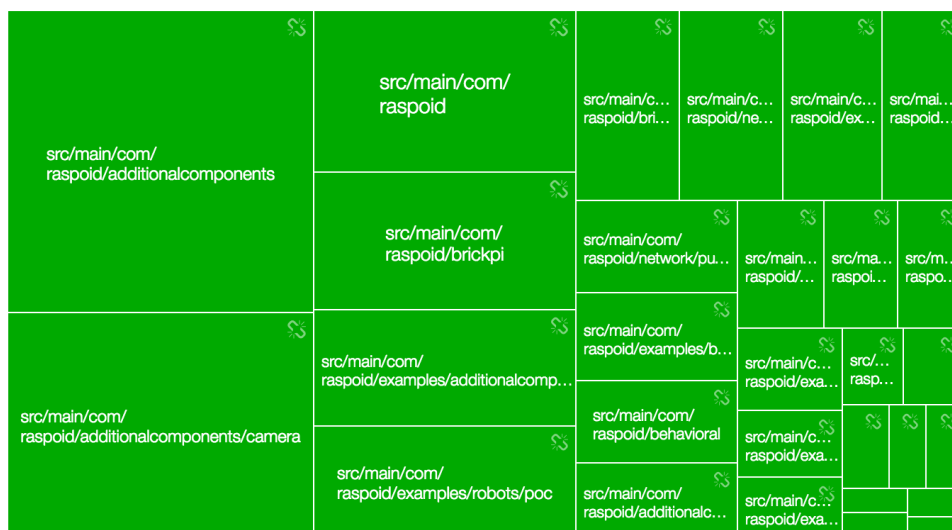


Figure 10.1: Treemap of public documented API - Block sizes: lines of codes in the corresponding package - Color: density of public documented API (%) (from red to green (green is better)).

Here are some metrics regarding the Java code of the Raspoid framework. Those metrics are only related to the Java code. We do not integrate here the code related to the Raspoid.com website (PHP, Symfony3, HTML5, CSS3 and Javascript) neither the code related to the accelerometer/gyroscope visualizers (Python, HTML5, CSS3 and Javascript).

¹Github.com: <https://github.com/> | Raspoid repository: <https://github.com/Raspoid/raspoid>

²<http://findbugs.sourceforge.net/>

Lines	21,028
Lines of code	7,063
Comment lines	4,958
Comments (%)	41.2%
Classes	182
Methods	681
Statements	2,946
Files	169
Directories	30
Public API	642
Public documented API (%)	100%
Issues	0
Reliability - Bugs	A - 0
Security - Vulnerabilities	A - 0
Duplication (%)	0%
Technical Debt	0

Table 10.1: Metrics

Here are some explanations about the meaning of those metrics. The definitions are directly taken from the SonarQube documentation³.

- **Lines:** Number of physical lines (number of carriage returns).
- **Lines of code:** Number of physical lines that contain at least one character which is neither a whitespace or a tabulation or part of a comment.
- **Comment lines:** Number of lines containing either comment or commented-out code. Non-significant comment lines (empty comment lines, comment lines containing only special characters, etc.) do not increase the number of comment lines.
- **Comments (%):** Density of comment lines = $\text{Comment lines} / (\text{Lines of code} + \text{Comment lines}) * 100$.
- **Classes:** Number of classes (including nested classes, interfaces, enumerations and annotations).
- **Methods:** Number of methods.
- **Statements:** Number of statements, without block definitions. Statements counter gets incremented by one each time a following keyword is encountered: if, else, while, do, for, switch, break, continue, return, throw, synchronized, catch, finally. Statements counter is not incremented by a class, method, field, annotation definition, package declaration and import declaration.
- **Files:** Number of files.
- **Directories:** Number of directories.
- **Public API:** Number of public Classes + number of public Functions + number of public Properties.
- **Public documented API (%):** Density of public documented API = $(\text{Public API} - \text{Public undocumented API}) / \text{Public API} * 100$.
- **Issues:** Number of issues.
- **Reliability Rating:** A = 0 Bug, B = at least 1 Minor Bug, C = at least 1 Major Bug, D = at least 1 Critical Bug, E = at least 1 Blocker Bug.
- **Security Rating:** A = 0 Vulnerability, B = at least 1 Minor Vulnerability, C = at least 1 Major Vulnerability, D = at least 1 Critical Vulnerability, E = at least 1 Blocker Vulnerability.
- **Duplication:** Density of duplication = $\text{Duplicated lines} / \text{Lines} * 100$. Duplicated lines = Number of lines involved in duplications. For a block of code to be considered as duplicated: there should be at least 10 successive and duplicated statements whatever the number of tokens and lines (differences in indentation as well as in string literals are ignored while detecting duplications).
- **Technical Debt:** Effort to fix all maintainability issues. This measure is expressed in minutes.

³<http://docs.sonarqube.org/display/SONAR/Metric+Definitions>

Part III

Additional Components - Tutorials

11.1 Operating principle

The aim of the ultrasound sensor is to detect obstacles in front of it, and to measure its distance to the sensor. Ultrasounds are sound waves with frequencies greater than 20 kHz, which is the upper limit of the human audible range. Our sensor operates at 40 kHz (HC-SR04).

The operating principle is based on reflection of sound waves: assuming that the speed of the wave is known, a measure of the time of flight gives the distance between them, because the distance is proportional to this time [24].

This operating principle is the same for the LEGO Mindstorms NXT ultrasonic sensor and for the HC-SR04 we present here. These sensors are composed of two parts: an ultrasonic emitter and an ultrasonic receiver. As presented in figure 11.1, an ultrasound wave is sent from the sender part of the sensor. If the wave encounters an obstacle, it is reflected. The receiver part of the sensor will then detect the reflected wave and use the time elapsed between sending and receiving waves to calculate the distance between the sensor and the first obstacle: $d = v.t/2$.

It is important to note that the accuracy of this kind of sensor is limited, because of physical properties of sound waves, and because of poor quality of the device:

- Ultrasound waves are propagated in three dimensions in the space, in a cone. The first object in the cone that the wave will encounter will reflect it. It is impossible to deduce with precision in which direction is the obstacle.
- The speed of the sound waves is sensible to air temperature and humidity, but the sensor assumes that this speed is a constant.
- The ultrasonic reflectance of the object is an important parameter and depends on the size of the object and its surface composition.
- Moreover, we have also noticed that the emission/detection are not as efficient in any direction, at least for small obstacles ; we deduce that each sensor should be calibrated individually to give more accurate results.

This type of sensor should only be used to evaluate proximity of obstacles, but no more and certainly not for accurate measuring.

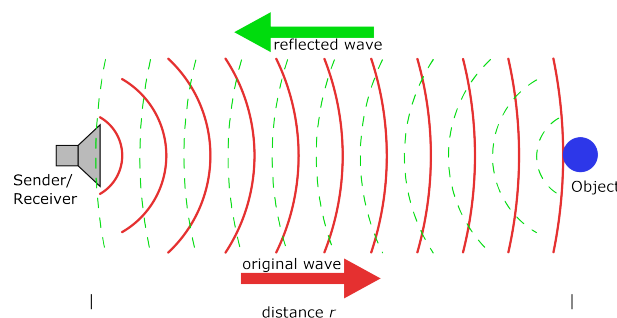


Figure 11.1: Ultrasonic sensor - Operating principle¹;

HC-SR04

The use of the HC-SR04 is rather simple. The sensor has four pins: one for the input voltage, one for the ground, and two pins (named "trig" and "echo") that will be connected to two GPIO pins on the Raspberry Pi.

As explained in the datasheet^[25] and presented on the time diagram in figure 11.2 (b), it is required to send a pulse with a width greater than 10 μs on the trig pin. The sensor will detect this request for a new measure, will produce a pulse with a frequency of 40 kHz, and will send it from the trig ultrasound emitter. The receiver part then waits for the reflected ultrasound signal. When detected, the sensor evaluates the time between sending and receiving ultrasound waves and returns a signal on the echo pin. This signal has a width corresponding to the evaluated distance.

The HC-SR04 is assumed to detect objects from 2 to 400 cm. We made some measurements to evaluate the accuracy of the sensor. These results are shown in figure 11.3 (right part). Orange circles refer to distances measured by the HC-SR04 sensor. The dashed blue line corresponds to actual distances. We can observe that measured distances are highly correlated with actual distances. We also show the error (multiplied by a factor of 10 to be readable) between measured and actual distances in red.

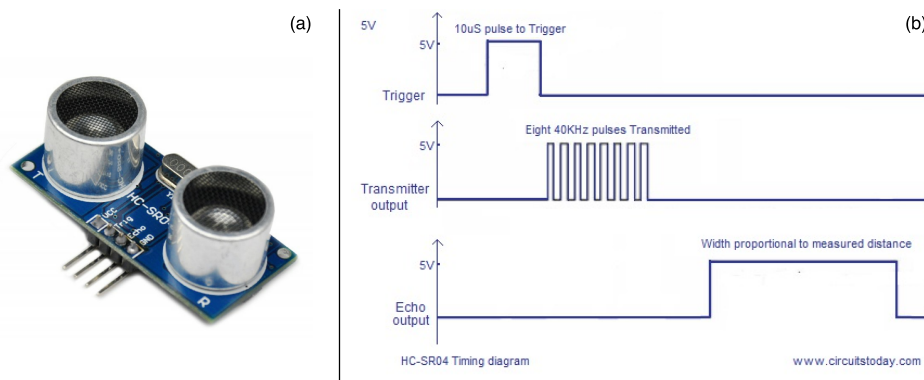


Figure 11.2: (a) HC-SR04 sensor - (b) timing diagram.

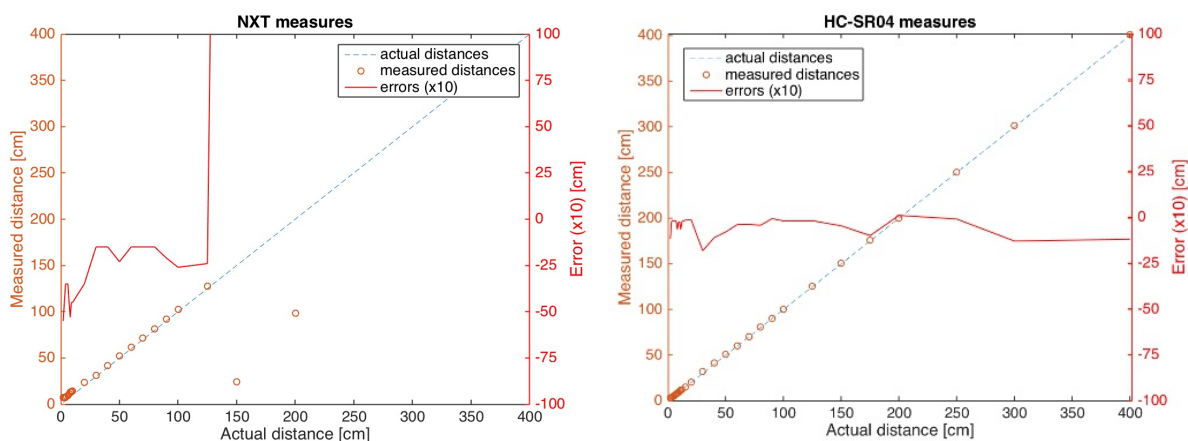


Figure 11.3: HC-SR04 / NXT - Measures comparison.

¹Source: https://commons.wikimedia.org/wiki/File:Sonar_Principle_EN.svg

LEGO Mindstorms NXT

The LEGO Mindstorms NXT 9846 ultrasonic sensor is assumed to detect objects from approximately 5 to 255 centimeters away (255 corresponding to the max 8-bit value). In figure 11.3 (left part), we can observe that measured distances are less accurate than with the cheaper HC-SR04 sensor. Moreover, distances measured above 125 cm no longer have any meaning.

11.2 Example of use with Raspoid

Circuit

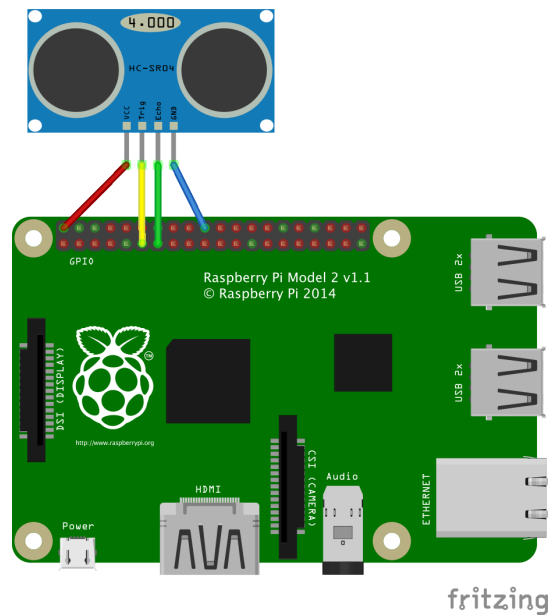


Figure 11.4: Ultrasonic sensor (HC-SR04) - Circuit

Program

This program example takes a new measure in cm from the HC-SR04 each 300 milliseconds, and prints this measure in the standard output. The trig pin is connected to GPIO pin 0 and the echo pin is connected to the GPIO pin 2, as presented in the figure above.

```

1 public static void main(String[] args) {
2     UltrasonicHCSR04 ultrasonicSensor = new UltrasonicHCSR04(GPIOPin.GPIO_00,
3     GPIOPin.GPIO_02);
4
5     while(true) {
6         Tools.log(ultrasonicSensor.getDistance() + "cm");
7         Tools.sleepMilliseconds(300);
8     }
9 }
```


I²C - Accelerometer & gyroscope

To illustrate the use of an I²C component with the Raspoid framework, we present the MPU6050, a chip combining an accelerometer and a gyroscope. This component is fascinating. The MPU6050 is mainly composed of a 3-axis gyroscope, a 3-axis accelerometer and six 16-bit analog-to-digital converters (ADCs): three for digitizing the gyroscope outputs and three for digitizing the accelerometer outputs ([26], p.7). The total dataset given by the MPU6050 includes 3-axis gyroscope and 3-axis accelerometer data.

The MPU6050 also provides a configurable digital low pass filter (DLPF). This onboard DLPF allows to remove noise coming from the accelerometer and from the gyroscope by keeping long-term changes, and filtering out short-term fluctuations. One way to do this is to weight changes over time and give little importance to new changes, by applying a formula like: $x = 0.96 * x + 0.04 * newx$. This onboard DLPF is particularly useful when using this kind of component with a Raspberry Pi. Indeed, the computation behind a DLPF is not that easy to perform and requires to get a lot of samples (in the order of 1000 measurements per second), which is only possible with real-time systems; this is not the case of a Raspberry Pi.

In the rest of this chapter, we will cover the operating principle of an accelerometer, the operating principle of a gyroscope, what they measure and how to combine accelerometer and gyroscope readings to obtain more accurate information (complementary filter).

These kind of components are called MEMS (MicroElectroMechanical Systems).

12.1 Operating principle

Accelerometer

An accelerometer measures "accelerations": force per unit of mass ($a = F/m$). With a 3-axis accelerometer, we can measure accelerations along the three orthogonal axis (x, y, z). Although measured values are often expressed using the g unit, all accelerations applied on the accelerometer are detected, whether due to gravity or to any other force applied to the component.

Unlike with the gyroscope, it is not necessary to track the values from the accelerometer over the time to determine the current orientation of the component at any given time. We just need a bit of trigonometry. On the other hand, values are often very noisy and only useful to determine angles over a "long" period of time.

While there exists many types of accelerometers, the operating principle of the accelerometer used on the MPU6050 rely on piezoelectricity [27](p.86). Imagine a cubic box containing a ball with a specific mass. Each face of the box is built with a piezoelectric crystal. When a piezoelectric crystal is compressed, it produces a voltage difference between its surfaces. This voltage difference can be amplified and measured. When the box is moved, the ball inside the box will act on the faces of the box, with a force depending on the acceleration applied to the box (Newton law). The piezoelectric crystal composing the face will then be compressed and an electric signal can then be detected and measured. Each pair of opposite faces composing the box are related to a specific axis.

Each accelerometer is characterized by a zero-g voltage level. It corresponds to the difference of voltage between surfaces of piezoelectric crystal composing a face of a box, when "0g" is applied to this face.

To evaluate the orientation of the component, each measured voltage is compared to this zero-g level. From those values in volts, we need to use the accelerometer sensitivity (expressed in mV/g) to get a value expressed in g . We now have a three component vector defining the acceleration applied on the component. It is just needed to use trigonometry to get angles between x, y and z axis and this force vector.

Gyroscope

A gyroscope measures a rate of rotation. It reads "zero" when stationary and reads positive or negative when rotating. As for the accelerometer, the gyroscope used on the MPU6050 rely on piezoelectricity. More precisely, this kind of gyroscope is called a "Coriolis vibrator gyroscope" (CVG, standardised by IEEE[28]). The rotation rate is determined from the Coriolis force induced by a vibrating structure placed inside the gyroscope supporting rotation. When a voltage is applied on a piezoelectric material, it is deformed. It is this principle that is used to vibrate the structure inside the gyroscope. When the gyroscope rotates, the vibrating structure inside will exert a force on its support. This force is the Coriolis force, exerted perpendicularly to the plan of vibration of the piezoelectric material and proportional to the rotational speed of the gyroscope. The assembly is made such that this force is exerted on another piezoelectric component, which distorted, will induce a voltage difference that can be amplified and measured.

Given that the gyroscope measures a rotation rate (degree per second (deg/s or dps)), an angle can only be determined by tracking this rotation rate over the time. It is then required to track this rate and integrate it along the time: $\Delta a = \Delta \tau / \Delta t$, with Δa the variation in angle, $\Delta \tau$ the variation of rate of rotation (in degree per second) and Δt the variation in time (in seconds). For a 3-axis gyroscope, three similar components are used: one for each orthogonal axis (x, y, z).

We can note that this tracking is rather tricky and may induce some drift problems. Indeed, the gyroscope is usually calibrated by the constructors, but when assembled on the printed circuit board (PCB), the zero-rate level and sensitivity may change. We need to calibrate the sensor by ourselves. To do this, when the gyroscope is powered-on and is stationary, we collect a series of samples during 5 seconds and the average of those values is used as the turn-on zero-rate level τ_0 . We then use the following formula to determine the rate of rotation from new values read on the MPU6050: $\tau_{actual} = \tau_{measured} - \tau_0$.

Unfortunately, this is not enough to avoid drift problem over a long period of time. Indeed, as presented in figure 12.1, there is always a noise on the signal from the gyroscope. When integrated over a long period of time, this noise may induce a large drift. The complementary filter presented in the next subsection will try to correct this problem.

Complementary filter

We implemented a complementary filter to merge data from the gyroscope and the accelerometer. The aim of this complementary filter is to use the best of the two worlds to approximate the orientation of the MPU6050 in the space, and to avoid too much noise on this estimation. The result is more stable and accurate.

The orientation of the MPU6050 is estimated for each direction with the following formula, with $\angle_t$ the estimated angle at time t, τ the rate of rotation read from the gyroscope and $\angle_{accel,t}$ the angle value read from the accelerometer at time t:

$$\angle_t = 0.96 * (\angle_{t-1} + \tau * dt) + 0.04 * \angle_{accel,t}$$

. To have a dt sufficiently small and avoid excessive errors when integrating the rotation rate read from the gyroscope, we use a thread which regularly updates this value of the estimated angle through the complementary filter.

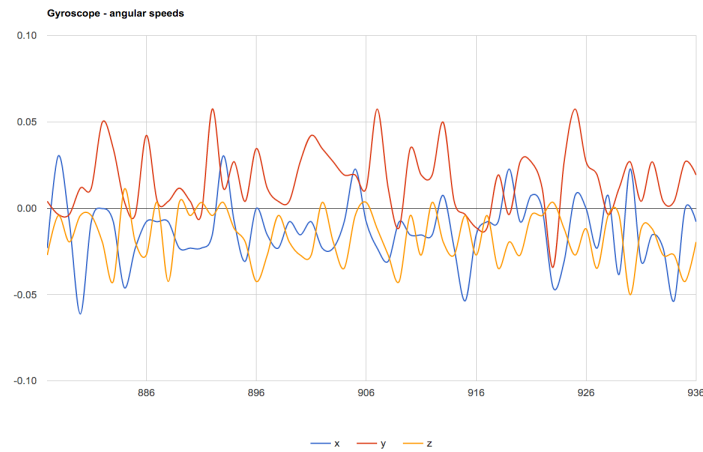


Figure 12.1: MPU6050 - Noise from the gyroscope, when stationary. dps on the vertical axis, seconds on the horizontal axis. Figure coming from our live data viewer for accelerometer/gyroscope values.

In the following code listing, we show our logic implemented to do so (we only show snippets related to the angle between the x axis and the component. Others are similar.).

```

1 private void updateValues() {
2     // Accelerometer
3     double[] accelerations = readScaledAccelerometerValues();
4     accelAccelerationX = accelerations[0];
5     ...
6     accelAngleX = getAccelXAngle(accelAccelerationX, accelAccelerationY,
7     accelAccelerationZ);
8     ...
9     // Gyroscope
10    double[] angularSpeeds = readScaledGyroscopeValues();
11    gyroAngularSpeedX = angularSpeeds[0] - gyroAngularSpeedOffsetX;
12    ...
13    // angular speed * time = angle
14    double dt = Math.abs(System.currentTimeMillis() - lastUpdateTime) / 1000.; // s
15    double deltaGyroAngleX = gyroAngularSpeedX * dt;
16    ...
17    lastUpdateTime = System.currentTimeMillis();
18
19    gyroAngleX += deltaGyroAngleX;
20    ...
21
22    // Complementary Filter
23    double alpha = 0.96;
24    filteredAngleX = alpha * (filteredAngleX + deltaGyroAngleX) + (1. - alpha) *
25    accelAngleX;
26    ...
27 }

```

Listing 12.1: Complementary filter to combine values from the accelerometer and the gyroscope.

12.2 Example of use with Raspoid

Circuit

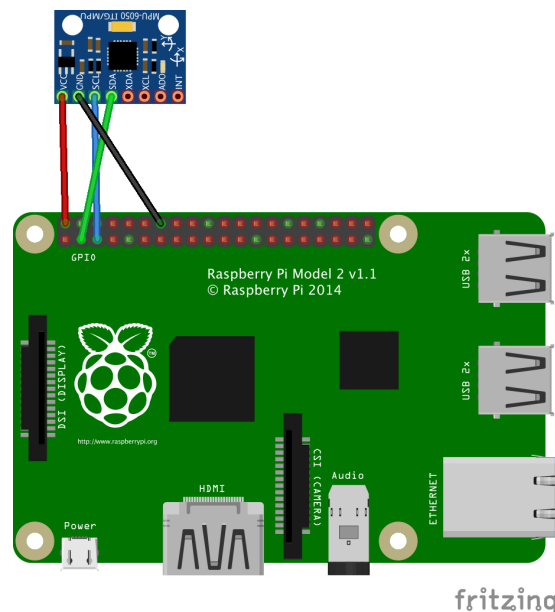


Figure 12.2: MPU6050 - Circuit

Program

In this example, we print the data retrieved from the accelerometer (accelerations & angles calculated from those accelerations), the values from the gyroscope (rates of rotation & angles calculated from those tracked rates of rotation) and the angles calculated through the complementary filter.

```

1 public static void main(String[] args) {
2     MPU6050 mpu6050 = new MPU6050();
3
4     while(true) {
5         Tools.log("_____");
6
7         // Accelerometer angles
8         Tools.log("Accelerometer:");
9         double[] accelAngles = mpu6050.getAccelAngles();
10        Tools.log("\t" + xyzValuesToString(angleToString(accelAngles[0]),
11            angleToString(accelAngles[1]), angleToString(accelAngles[2])));
12
13        double[] accelAccelerations = mpu6050.getAccelAccelerations();
14        Tools.log("\tAccelerations: " + xyzValuesToString(accelToString(
15            accelAccelerations[0]),
16            accelToString(accelAccelerations[1]), accelToString(
17            accelAccelerations[2])));
18
19        // Gyroscope angles
20        Tools.log("Gyroscope:");
21        double[] gyroAngles = mpu6050.getGyroAngles();
22        Tools.log("\t" + xyzValuesToString(angleToString(gyroAngles[0]),
23            angleToString(gyroAngles[1]), angleToString(gyroAngles[2])));
24
25        double[] gyroAngularSpeeds = mpu6050.getGyroAngularSpeeds();
26        Tools.log("\t" + xyzValuesToString(angularSpeedToString(gyroAngularSpeeds
27            [0]),

```

```
25         angularSpeedToString(gyroAngularSpeeds[1]), angularSpeedToString(  
26         gyroAngularSpeeds[2])));  
27     // Filtered angles  
28     Tools.log("Filtered angles:");  
29     double[] filteredAngles = mpu6050.getFilteredAngles();  
30     Tools.log("\t" + xyzValuesToString(angleToString(filteredAngles[0]),  
31         angleToString(filteredAngles[1]), angleToString(filteredAngles[2])))  
32     ;  
33     Tools.sleepMilliseconds(5);  
34 }  
35 }
```

Listing 12.2: MPU6050 - Example of use.

We can also note that we implemented a visualizer to easily present the data coming from an accelerometer, a gyroscope or a complementary filter. We present this visualizer in our Raspoid website (<http://www.raspoid.com>).

PWM - Servomotor

To illustrate the use of a component controlled with a PWM signal, we will present a servomotor. A servomotor is a motor suitable for use when one needs a precise control of the angular position of the rotor. They are really adapted for robotics.

These parts are really cheap (~5\$), and adapted for educational purposes and projects not requiring a lot of precision. When rotating, these servomotors rotate at their maximum speed (only with "ON/OFF" mode), and the position of the rotor is controlled via a resistive potentiometer, as presented in the next section. More sophisticated ones use optical rotary encoders to measure the speed of the rotor and are able to limit this speed to control the position of the rotor more quickly and with much more precision, by using a PID control algorithm (they control the speed of the rotor with regard to the remaining distance to travel, being increasingly slower).

13.1 Operating principle

Servomotors are connected with three wires: one for the ground, another for the voltage supply and the last one for the control signal. As a servomotor may draw a lot of current, it is highly recommended to use an external voltage supply.

To control the position of the rotor, the servomotors are controlled by sending pulses of variable widths on the control pin. The angle is determined by the duration of the high period of the pulse. For all the servomotors we worked with, they expect to see a pulse every 20 ms. The frequency of those pulses is therefore 50 Hz. But as we could observe, this frequency is not that important and may vary in a certain range. The most important point is the width of the high signal over a period. As an example, a duration of 1.5 ms should make the motor to turn to the 90° position (neutral position). Generally, the minimum pulse will be around 1 ms wide and the maximum pulse will be around 2 ms wide. We later provide an example to the reader.

As we can observe on the schema in figure 13.1 from the [29] paper, when the control signal is received by the servo, it is compared with a PWM signal generated by an internal signal generator inside the servo. The width of pulses generated by this internal generator is a function of the resistance of the potentiometer, itself controlled by the position of the rotor. With this mechanism of feedback loop, the rotor will be moved in the accurate direction until the target angle is reached.

When a servo is ordered to move, it will move as long as it receives the PWM signal. Once the destination position angle is reached, the rotor stops moving. Another characteristic of a servo is that when it has reached a position, it holds that position and it resists from moving out of that position. The maximum force that the servo can exert is the torque of the servo.

To illustrate our explanations regarding PWM signals sent to a servomotor, we took some measurements with an oscilloscope (a BitScope Micro¹). We show these measurements in figures 13.2 and 13.3. These are taken with a TowerPro MG90s[30] for which the range of available positions for the rotor is between 0 and 180 degrees. In figure 13.2, we can observe that the width of the PWM signal corresponding to the neutral position of the rotor (90°) is of about 1.7 ms. We can also observe that each pulse is repeated every 20 ms. In figure 13.3, we can respectively observe the widths of PWM signals corresponding to angles of 0 and 180 degrees. It is easy to interpolate the continuum of values between those min and max angles.

¹<http://bitscope.com/>

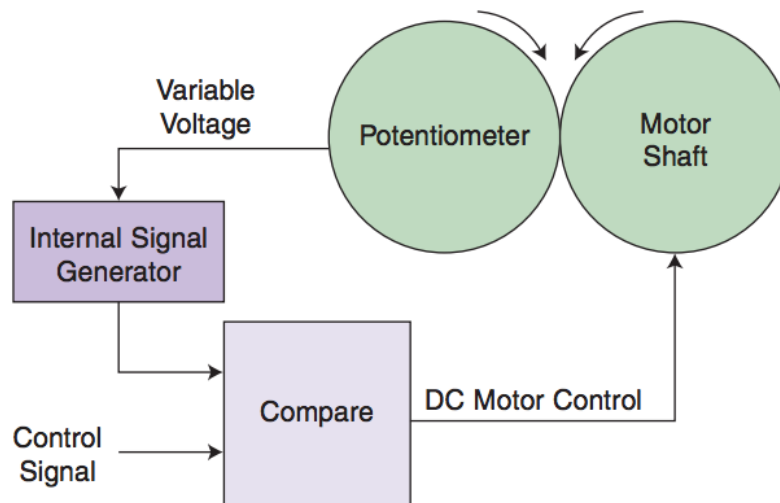


Figure 13.1: The inside of a servomotor[29].

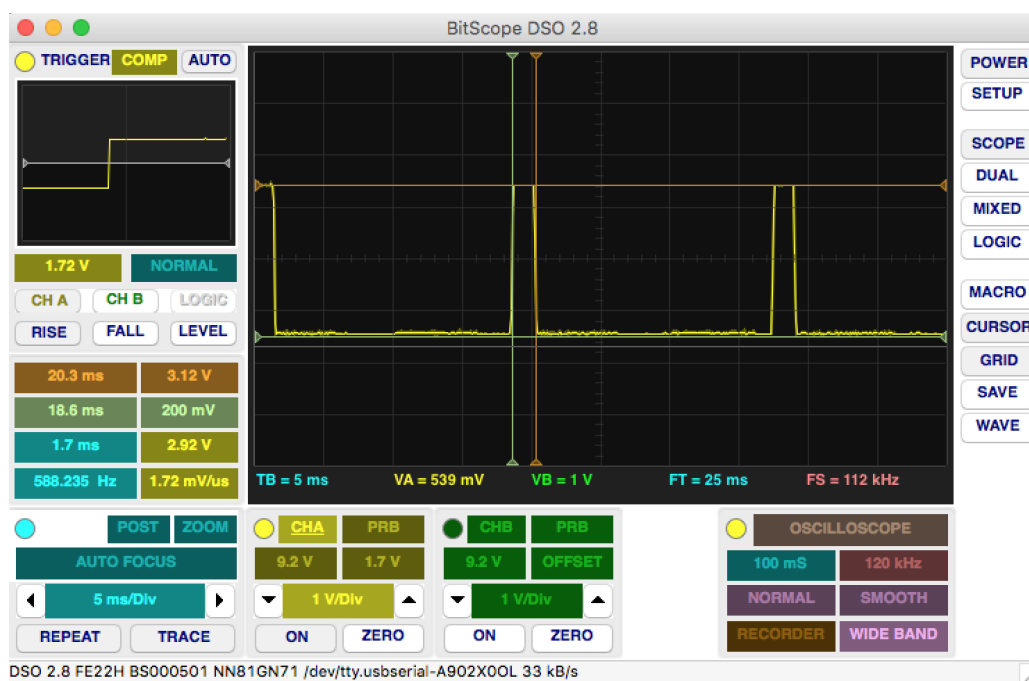


Figure 13.2: Servomotor - PWM signal for 90°.

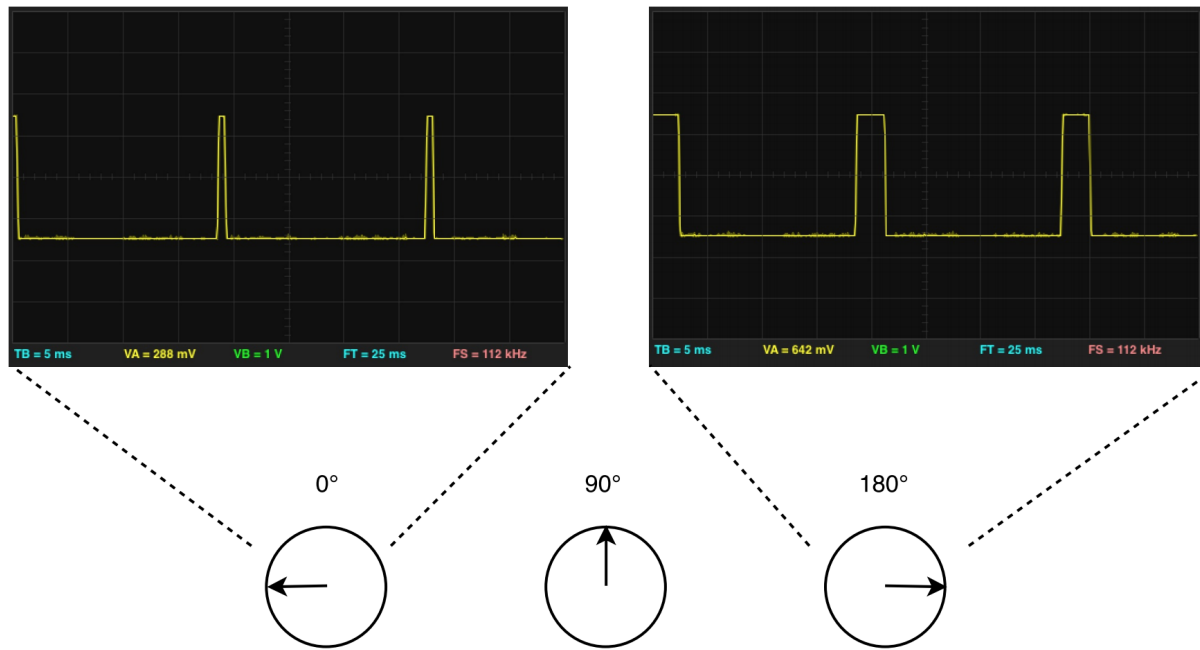


Figure 13.3: Servomotor - PWM signals w.r.t. angles.

13.2 Example of use with Raspoid

Circuit

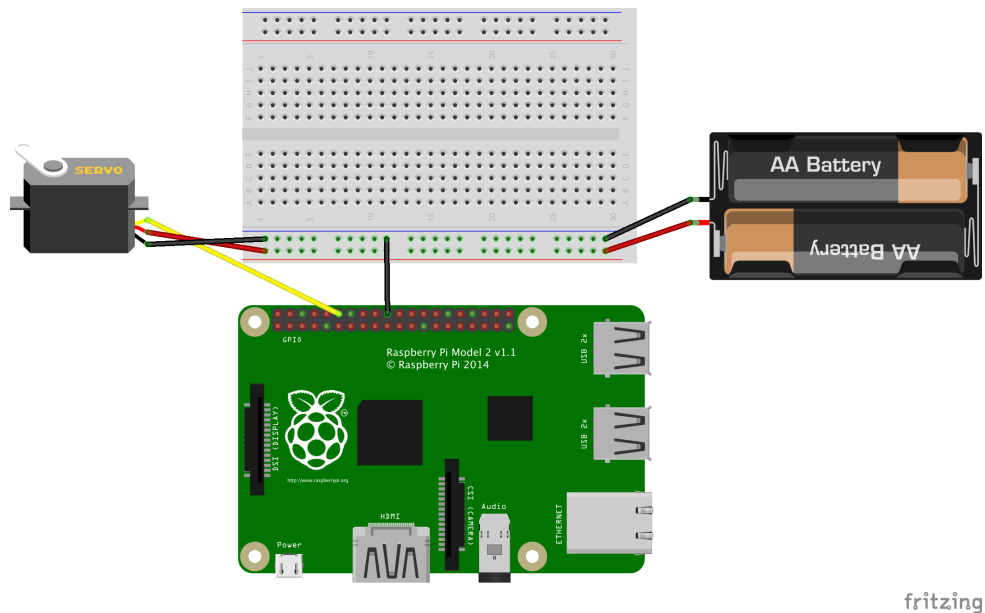


Figure 13.4: Servomotor - Circuit.

Program

As explained above, there are a lot a servos with different specifications such as the max angle of rotation, the rotation speed, the torque, etc. To easily control a servomotor with the Raspoid framework, we implemented a ServoMotor abstract class that can easily be extended for each existing servomotor. The only thing required is to give 6 specific parameters to the servo used. Those parameters can be found on the corresponding datasheets, or one can use our ServoMotorCalibration class to manually determine those coefficients. The setAngle method can then be applied on the servo to set a new position for the rotor. This method will send the corresponding PWM signal for a sufficient duration so that the motor rotates to the new position.

```
1 public static void main(String[] args) {
2     ServoMotor motor;
3
4     // Using a PWM pin
5     motor = new TowerProMG90S(PWMPin.PWM1);
6
7     // Using a PCA9685
8     //motor = new TowerProMG90S(new PCA9685(), PCA9685Channel.CHANNEL_01);
9
10    double[] angles = {0, 45, 90, 135, 180, 135, 90, 45, 0};
11
12    for(int i = 0; i < angles.length; i++) {
13        motor.setAngle(angles[i]);
14        Tools.log("position angle: " + motor.getPositionAngle());
15        Tools.sleepMilliseconds(1500);
16    }
17 }
```

Analog to digital - Photoresistor

14.1 Operating principle

Analog to Digital Converter (ADC)

Another type of useful sensors are sensors producing analog signals. As discussed earlier, the Raspberry Pi does not have any analog input pin. The GPIO pins are only able to decode digital signals. To use an analog sensor, we have to use an analog to digital converter (ADC). There exists a lot of ADCs. We will use here the PCF8591¹ from Philips, which is a cheap and widely used ADC. The PCF8591 is composed of 4 analog inputs and 1 analog output. The chip is connected to the Raspberry Pi using the I²C bus interface with the SDA and SCL pins. The only thing to do is to read the registers of the PCF8591 to get digital data corresponding to analog input signals received on the corresponding analog input pins of the chip. Each of the 4 analog inputs can independently measure voltages between two references voltages: V_{AGND} (analog ground) and V_{REF} (voltage reference input)[31] (p.4). The converter will return a digital value in the 0..255 range (unsigned byte), corresponding to 256 equal intervals between V_{AGND} and V_{REF} . An interesting information to note is that each PCF8591 has three address pins with which we can easily program the I²C hardware address. One can then connect up to 8 PCF8591 (2³) on the same I²C serial bus, and control up to 32 analog sensors (8 * 4) with only one Raspberry Pi².

As stated in [27] (p.197), the key characteristics of an ADC converter include accuracy, no-missing codes, resolution, conversion speed, and price. The PCF8591 is really cheap: about 1.80€ on the Farnell webshop³. The conversion speed is given by the maximum speed of the I²C bus and the converter makes use of the successive approximation conversion technique, with a resolution of 8 bits.

The successive approximation conversion technique is based on a dichotomy process. A Successive Approximation Register (SAR) is used with a Digital to Analogic Converter (DAC). The DAC produces an analog signal, proportional to the value stored in the SAR (a max value to start). This analog signal is then compared with the signal to convert. The result of the comparison is stored in the SAR, and the dichotomy process continues until completion. Those converters have conversion times of the order of tens of microseconds for resolutions of dozen of bits (90 µs in the case of PCF8591 for a resolution of 8 bits).

Photoresistor

To illustrate the use of an analog sensor with a Raspberry Pi and the Raspoid framework, we will show the utilization of a photoresistor⁴. A photoresistor is a resistor which resistance changes as a function of the intensity of incident light[27] (p.473). The detection is based on the band theory. The production of electron-hole is proportional to the intensity of the light, and the resistance is thus inversely proportional to the quantity of light (and can be considered as linear for a limited range of wave length). Cheap photoresistors for visible light uses cadmium sulfide (CdS) or cadmium selenide ($CdSe$)[27] (p.473).

¹Datasheet: <http://raspoid.com/download/datasheet/PCF8591>

²We tried this with 3 PCF8591 (12 analog inputs) (no more in stock), and it worked perfectly

³<http://be.farnell.com/fr-BE/nxp/pcf8591t-2-518/adc-single-8bit-11-1ksps-soic/dp/2400442RL> (April 2016)

⁴Other analog sensors have been integrated in the framework and tutorials are shown in the annexes of this report or directly from the Raspoid.com website.

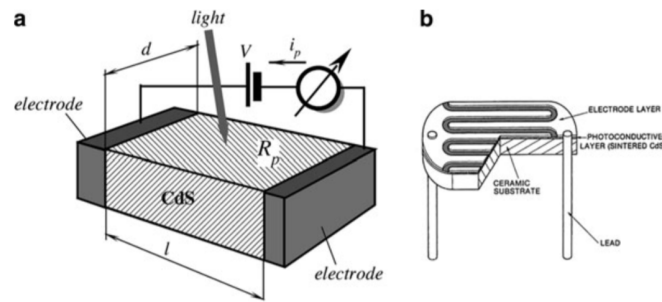


Figure 14.1: (Illustration from [27], p.473) Structure of a photoresistor (a) and a plastic-coated photoresistor having a serpentine shape (b)

14.2 Example of use with Raspoid

Circuit

In this experiment, we use a PCF8591 ADC, a photoresistor⁵ and a 1 k Ω resistor. The circuitry is shown in figure 14.2, and the corresponding code is shown in the listing 14.1.

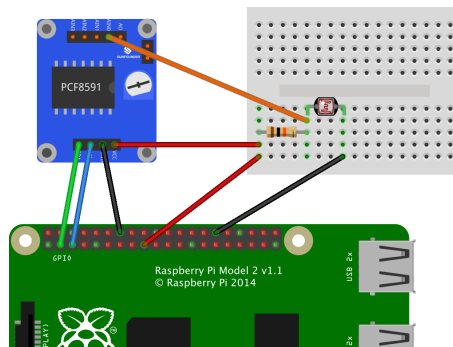


Figure 14.2: Photoresistor - Circuit

Program

This program example prints in the standard output each 250 milliseconds the light intensity detected by the photoresistor, in the 0..255 range. One can easily test this code and observe changes of values by varying light intensity on top of the photoresistor.

```

1 public static void main(String[] args) {
2     Photoresistor photoresistor = new Photoresistor(new PCF8591(),
3     PCF8591InputChannel.CHANNEL_0);
4
5     while(true) {
6         Tools.log(photoresistor.getIntensity());
7         Tools.sleepMilliseconds(250);
8     }
9 }

```

Listing 14.1: Photoresistor program example with Raspoid.

⁵For example, an LPRS N5AC501085 light dependent resistor at 2.12€ on the Farnell webshop (<http://be.farnell.com/fr-BE/lprs/n5ac501085/light-dependent-resistor-5mohm/dp/7482280>) (April 2016)

15.1 Raspberry Pi camera module

As explained in the introduction of part II of this report, the camera module distributed by the Raspberry Pi Foundation is able to take pictures and videos. The Foundation distributes two versions of the camera module: the "regular" and the "Pi NoIR". The Pi NoIR camera module works exactly like the regular one, with one difference: no infrared filter is used. This allows to conceive projects to "see in the dark" (while pictures and videos taken by daylight will look rather curious!).

In the Raspoid framework, we implemented a complete wrapper for the `raspistill` and `raspivid` command line tools distributed by the Raspberry Pi Foundation. With our java wrappers, one can easily launch a preview window on a HDMI or PCB display directly connected to the Raspberry Pi, take still photos and videos. We also developed a solution to easily stream live video from the Raspberry Pi through the network with GStreamer, and we compiled a complete version of OpenCV (Open-source Computer Vision) to be fully compatible with the Raspberry Pi (rather tricky). All the required tools and components are installed in our Raspoid OS image.

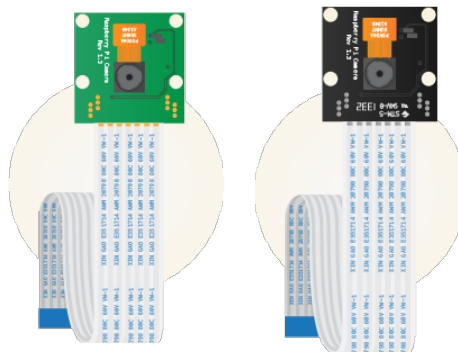


Figure 15.1: Camera modules - regular version on the left, NoIR version on the right

15.2 Example of use with Raspoid

To use the camera module, the first thing to do is to connect the camera to the CSI port of the Raspberry Pi, located behind the Ethernet port. The user then needs to enable the camera software by using the `raspi-config` command line tool (already done on the Raspoid OS image)¹

We will provide here one simple example for each category of features offered by the framework. It is impossible to list all the available methods here. We recommend the user to look at the complete API to see all the offered capabilities: <http://javadoc.raspoid.com/>.

The methods developed for this part of the framework are available in a static way. If the user needs to apply some specific modification on the image from the camera (vertical/horizontal split, width, height, opacity, resolution, exposure, etc.), we offer four configuration abstractions that can easily be used and

¹"\$ sudo raspi-config". Use the cursor keys to move to the camera option and select "enable". On exiting `raspi-config` it will ask to reboot. The enable option will ensure that on reboot the correct GPU firmware will be running (with the camera driver and tuning), and the GPU memory split is sufficient to allow the camera to acquire enough memory to run correctly.

then passed in argument of static methods. The `CameraControlOptions` refers to settings common to all images coming from the camera module: contrast, brightness, saturation, etc. The `PreviewConfig`, `PictureConfig` and `VideoConfig` entities are specific to the use of the corresponding features: width, height, output file name, quality, etc. Indeed, the range of available values for some options can be different for a picture, a preview or a video.

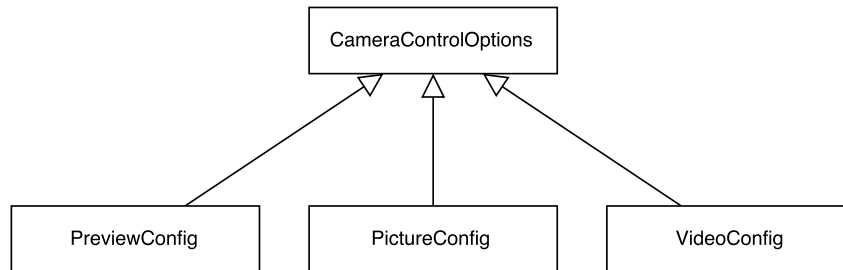


Figure 15.2: Camera Pi hierarchy configurations

Camera Preview

To take a preview from the camera, one first needs to connect the Raspberry Pi to an HDMI or PCB display. One can't simply use a VNC client to display a preview. It's a limitation from the `raspistill` command line tool. One can then execute the following example to see a preview of 5 seconds, with default parameters applied to the image.

```

1 public static void main(String[] args) {
2     // PREVIEW
3     CameraPi.preview(5000);
4 }

```

Listing 15.1: Camera module - Preview

Take still photographs

A picture can easily be taken with a simple command like the following. In the first example, the picture is saved with a default name. In the second example, the picture is saved with the "snowy_scenery.jpg" name, dimensions of 2592x1944 pixels and using the snow exposure mode.

```

1 public static void main(String[] args) {
2     // PICTURES
3     Picture picture1 = CameraPi.takePicture();
4     Tools.log("New picture: " + picture1.getFilePath());
5
6     PictureConfig pictureConfig = new PictureConfig("snowy_scenery", 2592, 1944);
7     pictureConfig.setExposureMode(ExposureMode.SNOW);
8     Picture picture2 = CameraPi.takePicture(pictureConfig);
9     Tools.log("New picture: " + picture2.getFilePath());
10 }

```

Listing 15.2: Camera module - Still photograph

Take videos

A video can be taken as easily as a picture. In the following example, we take a video of 5 seconds. By default, the created video is encoded in the .h264 format. We provide a method to easily convert the video to .mp4 video file, using the libav open-source video processing tool².

```
1 public static void main(String[] args) {
2     // VIDEOS
3     Video video = CameraPi.takeVideo(5000);
4     Tools.log("New video: " + video.getFilePath());
5     String convertedVideoFilePath = video.convertToMP4();
6     Tools.log("Converted file: " + convertedVideoFilePath);
7 }
```

Listing 15.3: Camera module - Take a video

Video streaming with GStreamer

We tried a lot of solutions to efficiently stream a video from the Raspberry Pi on the network. The most accurate solution is using the GStreamer tool³. Depending on width and height of the images chosen for the data incoming from the camera module, one can achieve really low latency (less than 100 ms).

In the following example, we simply set a GStreamer server to stream a video with a width of 640 pixels and an height of 360 pixels. The bitrate is set to 2500000 bits/second. The client (after installing the GStreamer client on its own) only needs to launch the command returned in the standard output once the GStreamer server is launched. We tried this on Linux and Mac OS X⁴. It works perfectly.

```
1 gst-launch-1.0 -v tcpclientsrc host="192.168.2.3" port="5000" ! gdpdepay !
   rtph264depay ! avdec_h264 ! videoconvert ! autovideosink sync=false"
```

Listing 15.4: Example of command to launch on the client side

```
1 public static void main(String[] args) {
2     // STREAMING
3     CameraPi.startGStreamerServer(NetworkUtilities.getIpAddresses().get(0),
4     NetworkUtilities.getAvailablePort(), 640, 360, true, true, 2500000, true
5     , false);
6 }
```

Listing 15.5: Camera module - Video streaming

Detect faces with OpenCV

We also compiled a specific version of OpenCV to test the tool with the Raspberry Pi camera module. We integrated this tool in the Raspoid framework. In the following example, we detect faces on pictures taken with the camera module. The possibilities offered by the OpenCV library are tremendous. This example is just for illustration. Further work with OpenCV could be promising.

²"sudo apt-get install -y libav-tools". Libav project website: <https://libav.org/>. Already installed on the Raspoid OS image.

³<https://gstreamer.freedesktop.org/>

⁴No reason it doesn't work on Windows. For compatibility reasons with the Raspberry Pi, the GStreamer server version (on the Raspberry Pi) must use the version 0.10 of the tool. On the client side, any version of the tool should be used (v1.8.1 at April 24, 2016.)

```
1 public static void main(String[] args) {  
2     // Load the OpenCV native library.  
3     System.loadLibrary(Core.NATIVE_LIBRARY_NAME);  
4  
5     // Take a picture with the camera pi  
6     String pictureId = new SimpleDateFormat("yyyy_MM_dd_HH_mm_ss").format(new Date()  
7     );  
8     PictureConfig pictureConfig = new PictureConfig("capture_" + pictureId, 640,  
9     480);  
10    pictureConfig.setVerticalFlip(true);  
11    String filePath = CameraPi.takePicture(pictureConfig).getFilePath();  
12  
13    // Look for faces  
14    Mat image = Highgui.imread(filePath);  
15    Rect[] faces = FaceDetector.detectFaces(image);  
16    Tools.log(String.format("%s faces detected", faces.length));  
17  
18    // Create a new picture, with detected faces  
19    FaceDetector.surroundFaces(image, faces, "output_" + pictureId + ".jpg");  
20 }
```

Listing 15.6: OpenCV - faces detection

Part IV

Educational Interest

Preamble

Since the beginning of eighties, a digital revolution is going on, which deeply modify our societies. This revolution has a characteristic: it is not only related to the work area as it was the case for the Industrial Revolution, but it also affects the private area. It occurred in several steps but very quickly: personal computers at affordable costs in the 80s, huge growth of the internet in the 90s, emergence of the smartphones in the 2000s, connected objects right now, etc.

In such a context, a gap is growing between those who know how to use a computer, and the others: we talk about the digital fracture. However, this breakage moves slowly to another area, between those who can simply use a computer, and those who know how to program it.

Education has a huge role to play in this field! Some countries have reached the milestone, and offer coding courses already at primary school. In our countries, this movement is still at the beginning, the awareness is gaining ground.

Often, a program uses a lot of abstractions, and it is tricky for young people. Learning coding by programming objects is probably more attractive!

Raspberry Pi in education

Since the last few years, some low-cost computing platforms appeared. Their success is unquestionable, and some of them such as the Raspberry Pi and Arduino are well known and widely used today.

The Raspberry Pi takes its essence in the education field. It was created by the the Raspberry Pi Foundation in 2012 with the aim of encouraging computer science learning in schools[32].

"The number of hobbyist programmers was dropping - and the number who where taking computer science to A Level was dropping, (...) One problem was teaching of ICT at school. Children were being exposed to learning about applications on a computer, rather than computer science as a discipline. (...) Over the past few decades there has been a huge change in the devices that were shipping. We went from things like the BBC Micro to black boxes used mainly for consuming content. We need to present computer science as a general purpose tool."
(Raspberry Pi Foundation co-founder Robert Mullins[32])

Developed to fill a gap in education, the project has been exploited by developers around the world and rapidly became a big success with more than 1 million items sold during the first year. Today, 8 million units have been sold, which makes it the best-selling computer in the UK. It is regularly enhanced: the brand new Raspberry Pi 3, presented in February 2016, is even more powerful and has the Wi-Fi and Bluetooth connectivity integrated on the board.

Success stories with low-cost platforms and in particular with Raspberry Pi in education are numerous. In the Department of Engineering and Automation Systems (DISA) of the University of Basque Country (UPV/EHU), teachers tried to find new approaches to increase the interest of students in their learning, and to improve teaching methods. They noticed an increasing interest from the students in the use of devices to solve any kind of engineering problems[33]. Programming practices tend to use more and more open-source projects, and the ability to deploy those solutions with cheap platform is particularly valuable. Today, this university incorporates different low-cost platforms into a variety of courses, and tries to improve even more the attractiveness of the offer for the students, by giving the opportunity to students to compete against each other, independently from the evaluation of the developed project. The teachers observed a significantly increased involvement of the student in comparison with previous editions. Moreover, the grades of those students were higher than previous years and higher than students who did not participate in the competition.

"By means of this low-cost platform, students experiment and learn lots of basics from several subject areas, even in topics independent of the course. (...) A very positive aspect is the fact that to improve the solutions implemented, students use knowledge they acquired in other subjects in the degree course." (Low-cost platforms used in Control Education: An educational case study[33])

Another success story of the Raspberry Pi in education is the Glyndwr/BCS Turing project, started in 2012 in the UK. The goal of this project, founded by the BCS (British Computer Society), was to introduce Welsh high-school students and staff to high-level programming and "computational thinking" [34], after decades of national neglect regarding Computer Science education. UK students had no notions of conventional Computer Science. The approach for the project is to give mobile workshops to

schools far away from the only two North Wales universities. To do this, it uses the LEGO NXT Mindstorms, the Raspberry Pi, and the PicoBoard. Each session starts with a presentation of the hardware, the software and a simple demonstration. The students then try to extend the function of the robot in a trivial manner, before diving into more challenging problems. A feedback was given by each student participating to the project: 82% among them said they had an increased interest in programming after the session, and 47% considered Computing and Technology as a hobby that they could develop during their free time.

In the Department of Computer Science of University of Almeria in Spain, the Raspberry Pi is used to fill the gap due to engineering courses too theoretical [35]. According to them, the main reason why courses stay theoretical is explained by two factors: the lack of time and the lack of practical resources. These shortcomings were overcome with the apparition of low-cost platforms and growing number of open-source projects. Today, they use a Raspberry Pi, and open-source software to perform simulations in a Matlab-like manner, using the Python programming language. The open-source SciPy library is used for mathematical algorithms and signal processing. NumPy and Matplotlib are used to work with vectors and draw graphics from these vectors as with Matlab. Even if this kind of simulations can be performed with classical simulation tools such as Matlab, Octave or others, they chose to work with a Raspberry Pi, Python and open-source libraries to show that it would be possible to perform all those simulations without any other additional computer than a Raspberry Pi (\$35). The use of those tools forces the students to install an operating system, to download and to install software using a package manager such as apt-get, etc. Even if those tasks seem trivial for computer science engineers, most engineers in other fields never learn about this during their studies. Moreover, the GPIO port allows to have physical communications with the outside world[35] to control cheap sensors and actuators. In this university, the aim is to introduce basic concepts for physics, mathematics, electronics and computer science with open-source tools, and a Raspberry Pi board.

We could give many other examples of use of a Raspberry Pi in education, including the diffusion of learning tools in developing countries, such as presented in the study [36]. In this paper, a complete modular computer is presented to be used in Uganda, where 37.7% of the population lives with 1.25\$ a day, partly because of a low education level. We could also point out projects created for young people such as Scratch¹, popularized with the Raspberry Pi, which allows to create programs like animations and games in a visual way (similar to what NXT-G proposes for LEGO Mindstorms programming).

The interest of students around the world for these new approaches using low-cost platforms and more didactic contents seems to be real and growing. The Raspberry Pi is an attractive device and we are convinced that this platform definitely has its place in education, at each learning step, including at EPL. For the price of a school book, students can have a complete kit to use a Raspberry Pi, which could be incorporated into a variety of courses. It could be used transversely between these courses, and improve the teaching methods in different areas such as:

- operating systems,
 - by installing and configuring multiple Linux operating systems such as Debian, Ubuntu, Fedora, Minix, etc.
 - by learning how to use a terminal,
 - by understanding file permissions and performing tasks with Superuser privileges,
 - by learning how to use apt-get, git and others to install and test open-source projects,
 - ...
- networking,
 - by using a Raspberry Pi as a router,
 - or as a DNS server, a Web server or any other kind of server,
 - by configuring network interfaces,
 - by configuring a TOR node,

¹<https://scratch.mit.edu/>

- ...
- security,
 - by sniffing packets on a network,
 - by using security tools with an operating system like Kali Linux,
 - ...
- low- and high-level programming,
 - by learning different programming languages such as Python, C, Java, etc.,
 - by replacing Matlab for a course like "Signals and systems" by open-source alternatives,
 - ...
- cloud computing,
 - by creating a cluster of Raspberry Pi,
 - by configuring a Docker installation,
 - ...
- electronics,
 - by using Raspoid to control a robot,
 - by using the GPIO port to control sensors and actuators,
 - by analyzing the Sense Hat² distributed by the Raspberry Pi Foundation and combining an accelerometer, a gyroscope, a joystick, etc.
 - ...
- etc.

As an example, a project like building a weather station with a Raspberry Pi at the core of the system could use transverse skills and courses such as electronics, networking, security, physics, etc.

The following argument is never highlighted in the literature, but another really interesting aspect of the Raspberry Pi is the use of micro-SD memory cards to host the operating system. This means that for only a few euros, a user can have multiple cards, with different operating systems. It is only needed to power off the Raspberry Pi, change the SD card, and to power on the Raspberry Pi again, to switch from an operating system to another. There is no need for multiple partitions on a computer, no need for virtual machines, etc. It is also possible to make a complete image of a SD card and restore this image on a SD card later. This can be used as a complete backup for an installation, but also to share a complete operating system containing specific data and configurations with other people. This is what we did with our Raspoid OS image. What we see here for the education field, is the opportunity for a teacher to configure a SD card with a specific installation (operating system and others) with specific needs for a course or a project, to create an image of this installation, and to share it with all his students. The student just need to download the image, flash it on a SD card, and their learning environment is ready. Each student has exactly the same installation: no need to support students under any version of Windows, Mac OS X and Linux. This kind of micro-SD card is really cheap³.

We also think that an investment in the Raspberry Pi field is a safe bet. There is a large and growing community of people creating and sharing software and hardware tools related to the Raspberry Pi. Possibilities are endless: with Raspoid we contribute to these by offering a Java framework leveraging the qualities of the Raspberry Pi, and adding the capabilities of Mindstorms and additional sensors and actuators.

The Raspberry Pi is a bit like a LEGO box offered to a child to teach him to develop his creativity, and to stimulate his reasoning, logic, and motivation for learning⁴.

²<https://www.raspberrypi.org/products/sense-hat/>

³<http://www.amazon.fr/dp/B00J2973JG>

⁴<http://granderecreation.com/les-avantages-du-jeu-de-construction-pour-les-enfants/>

Robotics in education

In the words of Kolb, "Learning is the process whereby knowledge is created through the transformation of experience" [37]. In his experiential learning model, Kolb identifies a learning cycle of four steps:

1. The concrete experience: performing some activity.
2. The reflective observation: perceive the effects of the activity.
3. The abstract conceptualization: appropriation of new generalized concepts in the light of the observed effects.
4. The active experimentation: experimenting again with the new knowledge.

From this perspective, learning is viewed as a cyclic process. This model highly applies for educational robotics and, most of the time, they are used as an experiential learning tool. Indeed, the development of a robot for tackling a problem fits naturally with this process: students create a robot with their current knowledge, then they analyze the quality of their artefact and they identify the remaining problems, afterwards they re-interpret their knowledge and integrate new concepts from observation, and finally they put into practice these newly acquired concepts by modifying their robot.

In his book [38], which incidentally gave its name to the LEGO Mindstorms kits, Papert also stresses the role of the robotics in the concretization and appropriation of the formal knowledge. According to him, robotics and more generally the computers, provide not only a way to learn by resolving problems by iterations, but also has a beneficial effect on the learning models developed by the students. Learning how to learn becomes a part of the problem for solving it more effectively.

While checking the literature, we found that many lectures already rely on educational robotics. The studies tend to qualitatively analyze the impact of introducing robotics education in the student's curriculum. We briefly report some of these studies here:

- In the paper of Williams [39], the Mindstorms kits were introduced in a computer engineering course to teach the student about C programming and embedded systems. The students were very favorable to this approach, and the author suggests that it is qualitatively effective to teach computer engineering.
- In the paper of Verner [40], a contest is used to learn engineering topics. The contest approach is used to emulate motivation and somehow transverse skills such as self-learning and research. The results provide quantitative measurements reporting that the majority of students made progress in their skills thanks to the contest. The impact on the motivation to learn science and engineering was evaluated positively by the majority of the students.
- In the recent review of Mubin [41], the authors perform an analysis of the robotics usage in education. It shows that some kits and frameworks similar to Raspoid, are used to support learning in the technical field (computer engineering and robotics) as well as in the science field (mathematics, geometry, kinetics). From the words of the authors, "Mindstorms robots have been shown to teach a wide array of subjects ranging from language, computer science/programming, physics, engineering design and robotics"[41].

These studies show that using educational robotics can be beneficial for students. However, the robots per se are not the miracle solution: it is rather the methodology used in parallel that allows for better results. In general, the methods used belong to the active methods which favor deeper learning and provide good results, even if it is not an absolute rule: in [42], the authors used robots to teach computer science, and measured a negative impact on the test scores of the students; Despite the interest of the students, an inappropriate organization, together with the lack of instructor's experience, did not allow to have good results.

Raspoid at EPL

Since 2000, the EPL faculty of the UCL implemented a PBL (Problem Based Learning) based curriculum, where learning takes advantage of a mix of projects and problems to be solved. This study [43] suggests that engineering students from the EPL acquired more skills by following the PBL curriculum than the students who followed the traditional curriculum. The initial experiments to integrate a pre-project in the curriculum included the realization of a mobile robot [44]. This led in particular to the conclusion that letting the students choose and/or build their own components was too complex. Therefore, a modular kit should be used with a high-level language like Java. Since then, the pre-project is given in the LFSAB1501 course, and it uses the LEGO Mindstorms kit. In our opinion, this project could benefit from some added value by using the Raspoid framework:

- The first advantage of the Raspoid framework is that it keeps a compatibility with the Mindstorms NXT components currently used for the project. From our experiments, the LEGO motors are the main asset of using the Mindstorms kit: they are easy to use, yet they remain very versatile. The sensors however (from the basic NXT kit) are less valuable. They can of course be used with Raspoid, but we found that the components available were limited compared to the plethora of sensors compatible with a Raspberry Pi. Moreover, what we call additional components, generally performed equally or better than the NXT sensors (e.g. the NXT ultrasonic sensor and the HC-SR04 sensor as discussed in chapter 11).
- As mentioned earlier, the Raspberry Pi has a great potential in the education field. Raspoid provides an implementation of a good set of commonly available components. The number and capabilities of these sensors greatly outnumbers the amount and capabilities of the Mindstorms NXT sensors (in the basic kit). The implementation provided can be used as is, or it can be used as a scaffolding for a more specific usage. We paid attention to extensively comment the implementation, therefore, it can also serve as an example to add other components to the framework. Finally, the components used are usually very cheap compared to Mindstorms components. For instance, the HCSR04 costs about 3€ in Amazon whereas the Ultrasonic NXT sensor costs more than ten times this price.
- In the PBL implementation at EPL, the project topics are semi-open. It allows to motivate the students while adequately delimiting the problem to avoid the over-specialization of the students[44]. Owing to the possibilities of Raspoid, we believe that it would permit a greater creativity and freedom in the projects, without necessarily increase their complexity. It is indeed the role of the tutor to lead the students in their research and developments. The tutor can perfectly define where to put the cursor on the intended complexity, which can range from using Raspoid as a black box, to re-implement a whole component for a specific purpose.

Subsequently, we present some project ideas with different difficulty levels (evaluated a 0-10 scale) that could be conducted at EPL.

18.0.1 Balancing robot

Audience: 1st year Q1 | **Difficulty:** 2-3 | **Estimated time:** 1 quadrimester

The idea is to propose a project where the students create a robot automatically balancing a tray as shown in the figure 18.1. A mass in a given range would be moved on the right side of the tray. The robot should then re-balance the tray as fast as possible when the balance is lost. The technique to re-balance

the tray is not imposed. The material involved would be at least a 6-axis gyroscope accelerometer such as the MPU6050, two LEGO motors, and a moving tray. At the end of the quadrimester, the students would compete against each other to assess the quality of their solution, with criteria such as speed and stability.

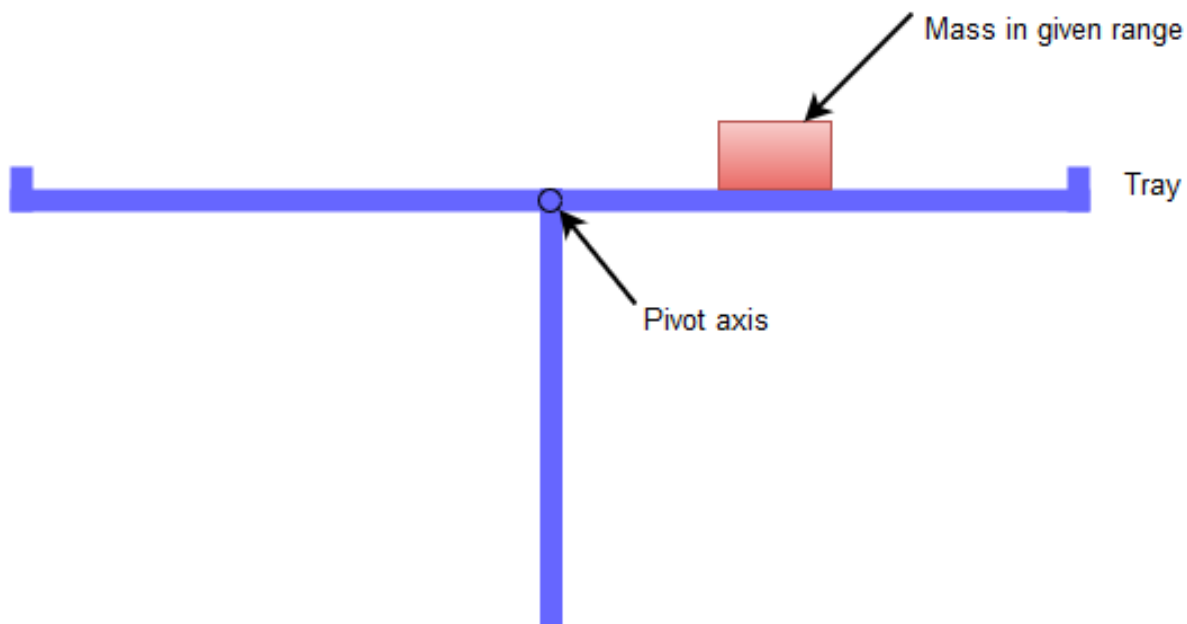


Figure 18.1: Balancing robot.

18.0.2 Digital metal detector

Audience: 1st year Q2 | **Difficulty:** 4-5 | **Estimated time:** 1 quadrimester

We propose to extend the metal detector project which was conducted in 2010 during the LFSAB1502 course. Each student group was given a minesweeper kit with an electronic board and components to build a metal detector. The project required that the metal detector should notify the user with a sound when a metallic object was detected. The sound had to be louder when the detection head was positioned closer to the object.

The idea proposed here would be to add an ADC component (like the PCF9685, integrated in the Raspoid framework), between the metal detector and a Raspberry Pi, in order to retrieve a digital value of the proximity with detected objects. The result could be showed on an LCD screen like the LCM1602 presented in appendix D.3. This project does not require the BrickPi component of Raspoid but only a Raspberry Pi with additional components.

18.0.3 BrickPi Motor Control

Audience: Master INFO | **Difficulty:** 7-8 | **Estimated time:** 2 Months

The goal of this project would be to implement a more accurate motor control functionality in the BrickPi firmware. Currently, the motor control is done on the Raspberry Pi. This means there is a delay between the time a position is measured, and the time a control decision, e.g. stopping the motor, is issued. That is why we implemented and used a PID controller to minimize the error when performing a movement.

The idea for the project would be to add a functionality in the BrickPi firmware to directly control the motor from the Atmega. The students would implement a new message specifying a specific power and

a target encoder value defining when the motor has to stop. The firmware would make sure this target encoder is not exceeded. This project would need a BrickPi, an FTDI cable to flash the firmware on the BrickPi and at least one NXT motor. At the end, some measurements could be taken and compared with the unmodified firmware solution.

18.0.4 Behavioral Programming

Audience: Master INFO / Difficulty: 4-5 / Estimated time: 1 Month

Behavioral programming is an interesting field for programming robots with a different paradigm. In Raspoid, we implemented a simple variant of this technique. The idea of this project would be to implement a more advanced version of the behavioral paradigm as described in [45]. In this approach, all the behaviors are executed at the same time (in parallel), and they use synchronization points to exchange messages. At each synchronization point, the b-thread specifies a set of messages requested, waited for or blocked. This scheme allows to express more scenarios and can potentially render the whole program more reactive since all the threads are running in parallel. For this project, no specific material is needed as all the behaviors could be virtual. Nonetheless, the behavioral paradigm is especially suited for autonomous robots. Therefore, we believe it would be interesting to let the students imagine their own autonomous robots with the behavioral approach: as an open subject, it should lead the students to be very creative.

18.0.5 Reliable Transfer Protocol

Audience: Master INFO / Difficulty: 6 / Estimated time: 1 Month

During the course LINGI2142, the students had to implement a reliable transfer protocol on top of UDP and IPv6. The students had to implement a window mechanism and the selective acknowledgment technique similarly to the SACK extension of TCP. The description of the protocol was given, in order to get an implementation compatible between the students. The idea for this project is to implement the reliable protocol as a Server in the network part of Raspoid. In order to assess that the communication works properly, a client could interact with the camera to retrieve an image or a video stream.

Conclusion

During the recent years, we had the opportunity to experiment with a Raspberry Pi, out of curiosity. Today, after one year of intensive work on this low-cost computer, we are convinced by its outstanding potential, not only as a hobby in the everyday life, but also in the field of education. In addition to its computer capabilities, the Raspberry Pi project is attractive because it gathers a large community around the world. This community is very active, and a lot of learning material, addressing various subjects, are available. We have shown that the Raspberry Pi could be used in a transverse manner in education, and that it is a good investment for students, particularly in the engineering field.

With the Raspoid framework, we offer a concrete solution to allow students to control electronics with the Java programming language, regardless of their initial skills. Usually, resources about the Raspberry Pi are presented for the Python programming language [46]. Java frameworks developed specifically for the Raspberry Pi are not so widespread. We believe that our work brings something new, and we hope that this contribution could be useful.

With our framework, it is possible to use a BrickPi to control LEGO Mindstorms sensors and motors. But it is much more: it is possible to use additional, out of the box, low-cost components such as an LCD display, an ultrasound sensor, a thermistor, some servomotors, a camera, some infrared components, and more. It is possible to use the network, to add communications in the developed projects, for instance with the Pushbullet services. And to make it more natural, we have implemented an additional layer to allow the use of the behavioral programming paradigm. All of this has been developed in a clear architecture, that can easily be extended by other developers, in the open-source spirit. All the needed resources for installation and use of Raspoid are available on the raspoid.com website.

Even if LEGO Mindstorms can be considered as a low-cost platform for robotics, our opinion as students is that it is still too costly. A LEGO Mindstorms base kit costs about 400€, and any additional sensor or motor costs between 25€ and 35€. LEGO Mindstorms are actually "Plug and Play", but the LEGO components are mainly used as "black boxes". For curious people, so for scientist students, it can be frustrating not to have control and understanding, especially on the underlying operating principle. Furthermore, the LEGO Mindstorms are much more limited than the Raspberry Pi, and are only restricted to robotics. In addition to being cheaper, the Raspberry Pi offers wider capabilities.

The main advantage that we found with LEGO Mindstorms is the ease of control and the versatility of the motors. However, there is no reason we could not integrate other servomotors with similar capabilities. This would mean that we would not need a BrickPi anymore: this is a trail for a future work.

Last, we decided to distribute our work under the LGPLv3 license¹. This allows everyone to freely use, distribute and modify our work, even for a commercial use. This license requires that derived works are licensed under the same license, while works that only use the framework as is do not fall under this restriction. This matches the idea that we have of the open-source. Open-source is really important from an educational viewpoint and we are confident that many people can bring value to the project. We will be happy to continue to maintain the framework, to wait for a feedback from the users and to help other developers to understand our work.

¹<http://www.gnu.org/licenses/lgpl-3.0.en.html>

Future work

During this project, we defined the scope of our work, and we stood there. However, we ruled out several ideas to improve the Raspoid framework. Still, we would like to share some development trails that could help to improve some technical and educational aspects of Raspoid.

- One trail could be to experiment the use of a real time kernel. This could potentially stabilize the communication jitter with the BrickPi. It could also allow to implement components using a communication protocol, which involves very precise timing. For instance, the DHT11 humidity sensor was not implemented because it was impossible to decode the received signals, which relies on a strict timing.
- As presented earlier, the Arduino UNO uses an Atmeg328 micro-controller to control its inputs/outputs (such as the BrickPi). These chips allow to perform some real-time controls. It could be great to integrate this Arduino board in the Raspoid framework, and take advantage of the power of this other well known learning tool.
- Another technical improvement, already exposed in the education part of this report, would be to modify the firmware of the BrickPi so that it directly controls the motor stop.
- This master thesis targeted the NXT Mindstorms only. Today, a new firmware was released for the BrickPi to support some EV3 components. It could be great to support these components in the framework.
- Another useful add-on, more intended for children, would be to write a scratch extension to make Raspoid usable with a visual programming language².
- As stated in the conclusion, it could be great to integrate new servomotors as additional components, eschewing the need of Mindstorms/BrickPi.
- In the artificial intelligence field, it could be possible to investigate the use of chatbots such as proposed with Pandorabots³. Combined with a microphone, it could then be possible to control a robot with natural language. As an example, one could drive a robot in a maze, by directing him with voice commands.

²https://wiki.scratch.mit.edu/wiki/Scratch_Extension, <https://github.com/LLK/scratchx/wiki>

³<http://www.pandorabots.com/>

Abbreviations

ADC	Analog-to-Digital Converter
ANSI	American National Standards Institute
API	Application Programming Interface
BCM	BroadCoM
BSD	Berkeley Software Distribution
CSI	Camera Serial Interface
CVG	Coriolis Vibrator Gyroscope
DAC	Digital-to-Analog Converter
DC	Direct Current
DLPF	Digital Low-Pass Filter
DNS	Domain Name System
DPI	Parallel Display Interface
GNU	GNU is Not Unix
GPCLK	General Purpose Clock
GPIO	General-Purpose Input/Output
GPL	GNU General Public License
I²C	Inter-Integrated Circuit
IDE	Integrated Development Environment
IEEE	Institute of Electrical and Electronics Engineers
IETF	Internet Engineering Task Force
IO	Input/Output
IR	InfraRed
ISP	In-System Programmer
JSON	JavaScript Object Notation
JTAG	Joint Test Action Group
LCD	Liquid Crystal Display
LED	Light Emitting Diode
LGPL	GNU Lesser General Public License

LSB Least Byte First

MEMS MicroElectroMechanical Systems

PBL Problem Based Learning

PCB Printed Circuit Board

PCM Pulse Code Modulation

PID Proportional Integral Derivative

PWM Pulse Width Modulation

RJ12 Registered Jack 12

RX Receiver

SACK Selective Acknowledgment

SAR Successive Approximation Register

SCL Serial Clock Line

SD Secure Digital

SDA Serial Data Line

SPI Serial Peripheral Interface

TX Transmitter

UART Universal Asynchronous Receiver/Transmitter

VNC Virtual Network Computing

VPS Virtual Private Server

List of Figures

1	NXT brick & Raspberry Pi 2.	1
1.1	NXT sensors (sound, light, touch and ultrasonic).	6
2.1	Raspberry Pi 2, Pinout.	8
3.1	BrickPi+ & Raspberry Pi.	11
3.2	BrickPi architecture.	13
3.3	Firmware and API interactions.	14
4.1	GANTT chart.	16
5.1	Raspoid base packages.	19
5.2	Dependencies	21
6.1	BrickPi API architecture.	26
6.2	BrickPi packet format	27
6.3	Class diagram: messages hierarchy.	29
6.4	PID - Tuning effect when increasing a coefficient.	31
6.5	Sensors hierarchy.	32
7.1	Additional components: parent classes.	33
7.2	GPIO components.	34
7.3	Adafruit 16-Channel 12-bit PWM/Servo Driver.	35
7.4	PWM components.	35
7.5	I ² C components.	36
7.6	Analog components	37
7.7	Camera Pi	38
8.1	Selection of a behavior by the arbitrator	39
9.1	Networking - Working principle.	41
9.2	Network - Main classes.	42
9.3	Threaded socket server.	44
9.4	Message-like socket server - Message format.	44
9.5	Pushbullet - Main classes.	46
10.1	Treemap of public documented API	47
11.1	Ultrasonic sensor - Operating principle.	51
11.2	HC-SR04 sensor & timing diagram.	52
11.3	HC-SR04 / NXT - Measures comparison.	52
11.4	Ultrasonic sensor (HC-SR04) - Circuit	53
12.1	MPU6050 - Noise from the gyroscope.	57
12.2	MPU6050 - Circuit	58
13.1	The inside of a servomotor.	62

13.2	Servomotor - PWM signal for 90°	62
13.3	Servomotor - PWM signals w.r.t. angles.	63
13.4	Servomotor - Circuit.	63
14.1	Structure of a photoresistor.	66
14.2	Photoresistor - Circuit	66
15.1	Camera Pi - Regular & NoIR.	67
15.2	Camera Pi hierarchy configurations	68
18.1	Balancing robot.	82
D.1	Example of 4 simple LEDs controlled with ON/OFF signals from the GPIO.	105
D.2	Example of a LED controlled with a PWM signal.	106
D.3	Example of a simple button connected to a GPIO pin.	108
D.4	Example of use of an LCM1602 - circuit.	109
D.5	Example of use of a joystick.	111
D.6	Example of use of a rotary encoder.	113
D.7	Example of use of a thermistor.	114
D.8	Example of use of a BMP180.	116
D.9	Basic Media Remote.	117
D.10	Example of use of an IR receiver and an IR transmitter.	118
D.11	Example of use of an ADXL345 accelerometer.	119
D.12	Example of use of a sound sensor.	120
D.13	Example of use of a buzzer (active or passive).	122
D.14	Example of use of a tracking sensor.	123
D.15	Example of use of an obstacle avoidance module.	124
E.1	Robot - Proof of concept - Overview.	127
E.2	MPU6050, passive buzzer, & PCA9685.	129
E.3	BMP180 & passive buzzer.	130
E.4	Raspberry Pi & BrickPi.	130
E.5	Raspberry Pi, BrickPi, Edimax WiFi, Camera Pi and ultrasonic sensor.	131
E.6	Battery holder LEGO idle wheel.	131
E.7	PCF8591, photoresistor, thermistor & sound sensor.	132
E.8	LCD display, infrared receiver & GPIO extension board.	132
E.9	Robot from the front & from the top.	132
E.10	Robot from behind & joystick remote.	133
F.1	BrickPi+ - Hardware schematics.	136
F.2	BrickPi - Hardware schematics.	137

Whole bibliography

- [1] Raffaele Grandi, Riccardo Falconi, and Claudio Melchiorri. “Robotic Competitions: Teaching Robotics and Real-Time Programming with LEGO Mindstorms”. In: *IFAC Proceedings Volumes* 47 (2014), 10598–10603.
- [2] LEGO MINDSTORMS Education. *[Manual] NXT User Guide*. LEGO. 2006.
- [3] LEGO. *[Manual] LEGO MINDSTORMS NXT Hardware Developer Kit*. 2006.
- [4] Roger A. Freedman Hugh D. Young. *University Physics with Modern Physics*. 13th. Addison-Wesley, 2011.
- [5] Gadgetoid. *[Online] Raspberry pinout*. 2016. URL: <http://pinout.xyz>.
- [6] eLinux. *[Online] RPi Low-level peripherals*. 2015. URL: http://elinux.org/RPi_Low-level_peripherals.
- [7] Michael McRoberts. *Beginning Arduino*. second. Apress, 2013.
- [8] Sparkfun (Jimb0). *[Online] Serial communication*. 2016. URL: <https://learn.sparkfun.com/tutorials/serial-communication>.
- [9] Sparkfun (SFUptownMaker). *[Online] I²C communication*. 2016. URL: <https://learn.sparkfun.com/tutorials/i2c>.
- [10] NXP Semiconductors. *[Datasheet] I²C-bus specification and user manual*. 2014.
- [11] Jack Creasey. *Raspberry Pi Essentials*. Packt Publishing Ltd, 2015.
- [12] Gordon Henderson. *[Online] Software PWM*. 2016. URL: <http://pi4j.com/apidocs/com/pi4j/wiringpi/SoftPwm.html>.
- [13] MIPI Alliance. *[Online] CSI specifications*. 2016. URL: <http://mipi.org/specifications/camera-interface>.
- [14] Dexter Industries. *[Online] LEGO MINDSTORMS Motors with Raspberry Pi (BrickPi 0.1)*. 2013. URL: <http://www.dexterindustries.com/howto/lego-mindstorms-motors-with-raspberry-pi-brickpi-0-1/>.
- [15] Toshiba. *[Datasheet] TB6612FNG Driver IC for Dual DC motor*. 2007.
- [16] Atmel. *[Datasheet] ATmega48A/PA/88A/PA/168A/PA/328/P*. 2015.
- [17] Microchip. *[Datasheet] MCP3021*. 2013.
- [18] Raspberry Pi Foundation. *[Online] BCM2836*. URL: <https://www.raspberrypi.org/documentation/hardware/raspberrypi/bcm2836/README.md>.
- [19] Eben Upton. *[Online] Raspberry Pi 3 on sale*. 2016. URL: <https://www.raspberrypi.org/blog/raspberry-pi-3-on-sale/>.
- [20] NXP. *[Datasheet] PCA9685 - 16-channel, 12-bit PWM Fm+ I²C-bus controller*. 2015.
- [21] Raspberry Pi Foundation. *[Online] Camera - Raspberry Pi*. 2016. URL: <https://www.raspberrypi.org/documentation/hardware/camera.md>.
- [22] David Harel, Assaf Marron, and Gera Weiss. “Behavioral programming”. In: *Communications of the ACM* 55.7 (2012), pp. 90–100.
- [23] leJOS. *[Online] leJOS Behavioral Programming*. 2016. URL: <http://www.lejos.org/nxt/nxj/tutorial/Behaviors/BehaviorProgramming.htm>.
- [24] Clarence W. de Silva. *Sensors and Actuators: Engineering System Instrumentation*. second. CRC Press, 2015.
- [25] Cytron Technologies. *[Datasheet] HCSR04 Ultrasonic Sensor*. 2013.

- [26] InvenSense. *[Datasheet] MPU-6000 and MPU-6050, Register Map and Descriptions, Revision 4.2*. 2013.
- [27] Jacob Fraden. *Handbook of Modern Sensors, Physics, Design, and Applications*. Fourth Edition. Springer, 2010.
- [28] *[Standard] IEEE Standard Specification Format Guide and Test Procedure for Coriolis Vibratory Gyros*. Aerospace and Electronic Systems Society, 2004.
- [29] Nathaniel Pinckney. “Pulse-width modulation for microcontroller servo control”. In: *IEEE Potentials* (2006).
- [30] TowerPro. *[Datasheet] MG90S Metal Gear Servo*.
- [31] Philips. *[Datasheet] PCF8591 - 8-bit A/D and D/A converter*. 2013.
- [32] Chris Edwards. “Not-so-humble Raspberry Pi gets big ideas”. In: *Engineering & Technology* 8.3 (2013), pp. 30–33.
- [33] R. Priego E. Irigoyen E. Larzabal. “Low-cost platforms used in Control Education: An educational case study”. In: *Advances in Control Education*. The International Federation of Automatic Control. 2013.
- [34] Nigel Houlden Vic Grout. “Taking Computer Science and Programming into Schools: The Glyn-dwr/BCS Turing Project”. In: *Procedia - Social and Behavioral Sciences* 141 (2014), pp. 680–685.
- [35] Ángeles Hoyo et al. “Teaching Control Engineering Concepts using Open-Source tools on a Raspberry Pi board”. In: *IFAC Workshop on Internet Based Control Education IBCE15 — Brescia* 48.29 (2015), pp. 99–104.
- [36] Nof Nasser Eddin Ben Falconer Colin Oram Murat Ali Jozef Hubertus Alfonsus Vlaskamp. “Technical development and socioeconomic implications of the Raspberry Pi as a learning tool in developing countries”. In: *Computer Science and Electronic Engineering Conference (CEEC)* 5 (2013).
- [37] David A Kolb. *Experiential learning: Experience as the source of learning and development*. FT press, 2014.
- [38] Seymour Papert. *Mindstorms: Children, computers, and powerful ideas*. Basic Books, Inc., 1980.
- [39] Andrew B Williams. “The qualitative impact of using LEGO MINDSTORMS robots to teach computer engineering”. In: *Education, IEEE Transactions on* 46.1 (2003), p. 206.
- [40] Igor M Verner and David J Ahlgren. “Robot contest as a laboratory for experiential engineering education”. In: *Journal on Educational Resources in Computing (JERIC)* 4.2 (2004), p. 2.
- [41] Omar Mubin et al. “A review of the applicability of robots in education”. In: *Journal of Technology in Education and Learning* 1 (2013), pp. 209–0015.
- [42] Barry Fagin and Laurence Merkle. “Measuring the effectiveness of robots in teaching computer science”. In: *ACM SIGCSE Bulletin* 35.1 (2003), pp. 307–311.
- [43] Benoît Galand, Mariane Frenay, Benoît Raucent, et al. “Effectiveness of problem-based learning in engineering education: a comparative study on three levels of knowledge structure”. In: *International Journal of Engineering Education* 28.4 (2012), pp. 939–947.
- [44] Edurne Aguirre and Benoît Raucent. “L’apprentissage par projet... Vous avez dit projet? Non, par projet”. In: *19ème colloque de l’Association Internationale de Pédagogie Universitaire (AIPU), Louvain-la-Neuve-Belgique* 29 (2002).
- [45] David Harel et al. “Behavioral programming, decentralized control, and multiple time scales”. In: *Proceedings of the compilation of the co-located workshops on DSM’11, TMC’11, AGERE! 2011, AOOPEs’11, NEAT’11, & VMIL’11* (2011), pp. 171–182.
- [46] Eben Upton. *[Online] PyPy on Pi*. 2013. URL: <https://www.raspberrypi.org/blog/pypy-on-pi/>.
- [47] InvenSense. *[Datasheet] MPU-6000 and MPU-6050 Product Specification Revision 3.4*. 2013.

Articles only

- [1] Raffaele Grandi, Riccardo Falconi, and Claudio Melchiorri. “Robotic Competitions: Teaching Robotics and Real-Time Programming with LEGO Mindstorms”. In: *IFAC Proceedings Volumes* 47 (2014), 10598–10603.
- [22] David Harel, Assaf Marron, and Gera Weiss. “Behavioral programming”. In: *Communications of the ACM* 55.7 (2012), pp. 90–100.
- [29] Nathaniel Pinckney. “Pulse-width modulation for microcontroller servo control”. In: *IEEE Potentials* (2006).
- [32] Chris Edwards. “Not-so-humble Raspberry Pi gets big ideas”. In: *Engineering & Technology* 8.3 (2013), pp. 30–33.
- [34] Nigel Houlden Vic Grout. “Taking Computer Science and Programming into Schools: The Glyn-dwr/BCS Turing Project”. In: *Procedia - Social and Behavioral Sciences* 141 (2014), pp. 680–685.
- [35] Ángeles Hoyo et al. “Teaching Control Engineering Concepts using Open-Source tools on a Raspberry Pi board”. In: *IFAC Workshop on Internet Based Control Education IBCE15 — Brescia* 48.29 (2015), pp. 99–104.
- [36] Nof Nasser Eddin Ben Falconer Colin Oram Murat Ali Jozef Hubertus Alfonsus Vlaskamp. “Technical development and socioeconomic implications of the Raspberry Pi as a learning tool in developing countries”. In: *Computer Science and Electronic Engineering Conference (CEEC)* 5 (2013).
- [39] Andrew B Williams. “The qualitative impact of using LEGO MINDSTORMS robots to teach computer engineering”. In: *Education, IEEE Transactions on* 46.1 (2003), p. 206.
- [40] Igor M Verner and David J Ahlgren. “Robot contest as a laboratory for experiential engineering education”. In: *Journal on Educational Resources in Computing (JERIC)* 4.2 (2004), p. 2.
- [41] Omar Mubin et al. “A review of the applicability of robots in education”. In: *Journal of Technology in Education and Learning* 1 (2013), pp. 209–0015.
- [42] Barry Fagin and Laurence Merkle. “Measuring the effectiveness of robots in teaching computer science”. In: *ACM SIGCSE Bulletin* 35.1 (2003), pp. 307–311.
- [43] Benoît Galand, Mariane Frenay, Benoît Raucent, et al. “Effectiveness of problem-based learning in engineering education: a comparative study on three levels of knowledge structure”. In: *International Journal of Engineering Education* 28.4 (2012), pp. 939–947.
- [44] Edurne Aguirre and Benoît Raucent. “L’apprentissage par projet... Vous avez dit projet? Non, par projet”. In: *19ème colloque de l’Association Internationale de Pédagogie Universitaire (AIPU), Louvain-la-Neuve-Belgique* 29 (2002).
- [45] David Harel et al. “Behavioral programming, decentralized control, and multiple time scales”. In: *Proceedings of the compilation of the co-located workshops on DSM’11, TMC’11, AGERE! 2011, AOOPEs’11, NEAT’11, & VMIL’11* (2011), pp. 171–182.

Books only

- [4] Roger A. Freedman Hugh D. Young. *University Physics with Modern Physics*. 13th. Addison-Wesley, 2011.
- [7] Michael McRoberts. *Beginning Arduino*. second. Apress, 2013.
- [11] Jack Creasey. *Raspberry Pi Essentials*. Packt Publishing Ltd, 2015.
- [24] Clarence W. de Silva. *Sensors and Actuators: Engineering System Instrumentation*. second. CRC Press, 2015.
- [27] Jacob Fraden. *Handbook of Modern Sensors, Physics, Design, and Applications*. Fourth Edition. Springer, 2010.

- [37] David A Kolb. *Experiential learning: Experience as the source of learning and development*. FT press, 2014.
- [38] Seymour Papert. *Mindstorms: Children, computers, and powerful ideas*. Basic Books, Inc., 1980.

URLs only

- [5] Gadgetoid. *[Online] Raspberry pinout*. 2016. URL: <http://pinout.xyz>.
- [6] eLinux. *[Online] RPi Low-level peripherals*. 2015. URL: http://elinux.org/RPi_Low-level_peripherals.
- [8] Sparkfun (Jimb0). *[Online] Serial communication*. 2016. URL: <https://learn.sparkfun.com/tutorials/serial-communication>.
- [9] Sparkfun (SFUptownMaker). *[Online] I²C communication*. 2016. URL: <https://learn.sparkfun.com/tutorials/i2c>.
- [12] Gordon Henderson. *[Online] Software PWM*. 2016. URL: <http://pi4j.com/apidocs/com/pi4j/wiringpi/SoftPwm.html>.
- [13] MIPI Alliance. *[Online] CSI specifications*. 2016. URL: <http://mipi.org/specifications/camera-interface>.
- [14] Dexter Industries. *[Online] LEGO MINDSTORMS Motors with Raspberry Pi (BrickPi 0.1)*. 2013. URL: <http://www.dexterindustries.com/howto/lego-mindstorms-motors-with-raspberry-pi-brickpi-0-1/>.
- [18] Raspberry Pi Foundation. *[Online] BCM2836*. URL: <https://www.raspberrypi.org/documentation/hardware/raspberrypi/bcm2836/README.md>.
- [19] Eben Upton. *[Online] Raspberry Pi 3 on sale*. 2016. URL: <https://www.raspberrypi.org/blog/raspberry-pi-3-on-sale/>.
- [21] Raspberry Pi Foundation. *[Online] Camera - Raspberry Pi*. 2016. URL: <https://www.raspberrypi.org/documentation/hardware/camera.md>.
- [23] leJOS. *[Online] leJOS Behavioral Programming*. 2016. URL: <http://www.lejos.org/nxt/nxj/tutorial/Behaviors/BehaviorProgramming.htm>.
- [46] Eben Upton. *[Online] PyPy on Pi*. 2013. URL: <https://www.raspberrypi.org/blog/pypy-on-pi/>.

Datasheets only

- [10] NXP Semiconductors. *[Datasheet] I²C-bus specification and user manual*. 2014.
- [15] Toshiba. *[Datasheet] TB6612FNG Driver IC for Dual DC motor*. 2007.
- [16] Atmel. *[Datasheet] ATmega48A/PA/88A/PA/168A/PA/328P*. 2015.
- [17] Microchip. *[Datasheet] MCP3021*. 2013.
- [20] NXP. *[Datasheet] PCA9685 - 16-channel, 12-bit PWM Fm+ I²C-bus controller*. 2015.
- [25] Cytron Technologies. *[Datasheet] HCSR04 Ultrasonic Sensor*. 2013.
- [26] InvenSense. *[Datasheet] MPU-6000 and MPU-6050, Register Map and Descriptions, Revision 4.2*. 2013.
- [30] TowerPro. *[Datasheet] MG90S Metal Gear Servo*.
- [31] Philips. *[Datasheet] PCF8591 - 8-bit A/D and D/A converter*. 2013.
- [47] InvenSense. *[Datasheet] MPU-6000 and MPU-6050 Product Specification Revision 3.4*. 2013.

