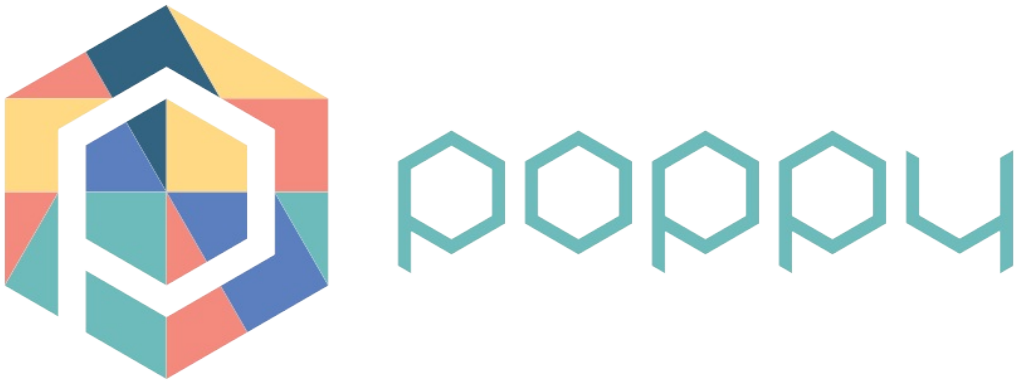
Poppy Documentation

release 1.1.0

included !

PyPot

Table of Contents



Introduction

About

Welcome to the manual of the Poppy Project, an open-source robotic platform.

In this documentation, we will try to cover everything from the short overview of what is possible within the project to the details on how you can build a Poppy robot or reproduce one of the pedagogical activity.

In the first chapter, we will give you a simple but exhaustive overview of what you can do within this project, so you can quickly focus on the following chapters that cover what you are really interested in. While some advanced chapters may require a good understanding of mechanics, electronics or computer sciences, the Getting Started section is intended to be easily accessible by all readers.

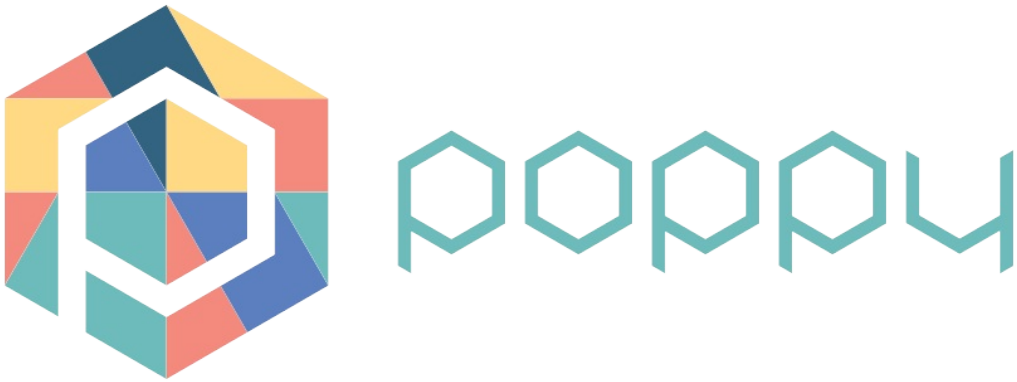
Version

This is the version 1.0 of the Poppy documentation, updated at Thu Aug 18 2016 10:51:19 GMT+0000 (UTC).

It contains the documentation for:

- Poppy Humanoid hardware version 1.0.1
- Poppy Torso hardware version 1.0.1
- Poppy ErgoJr hardware version beta 6
- poppy_humanoid library version 1.1.1
- poppy_torso library version 1.1.5
- poppy_ergo_jr library version 1.4.0
- poppy.creatures library version 1.7.1
- pypot library version 2.10.0

Getting Started



Poppy Project is an open-source platform for the creation, use and sharing of interactive 3D printed robots. It gathers an interdisciplinary community of beginners and experts, scientists, educators, developers and artists. They all share a vision: robots are powerful tools to learn and be creative and collaborate to improve the project. They develop new robotic behaviors, create pedagogical contents, design artistic performances, improve the software or even create new robots.

The Poppy community develops robotic creations that are easy to build, customize, and deploy. We promote open-source by sharing hardware, software. A web platform is associated enabling the community to share experiences and to contribute to its improvement.

To ease these exchanges two supports are available:

- The poppy-project forum for help, discussions and sharing ideas.
- GitHub to submit your contributions.

All sources of the Poppy Project (software and hardware) are available on GitHub.

The Poppy project has been originally designed at Inria Flowers.

The Poppy creatures

Poppy creatures are open-source robots, available for download and modification (Creative Commons Attribution-ShareAlike for the hardware and GPLv3 for the software). They were designed with the same principles in mind.

All Poppy creatures:

- are made from pieces of printable 3D and Dynamixel motors,
- use an embedded board for control (a Raspberry Pi 2 or Odroid for older versions),
- are based on a Python library, pypot, allowing to control Dynamixel servomotors in an easy way,
- have a simulated version available (based on V-REP),
- can be controlled using a visual programming language (Snap! a variation of Scratch) and a textual language Python. They are also programmable through a REST API, which enables the control with other programming language,
- come with associated documentation, tutorials, examples, pedagogical activities.

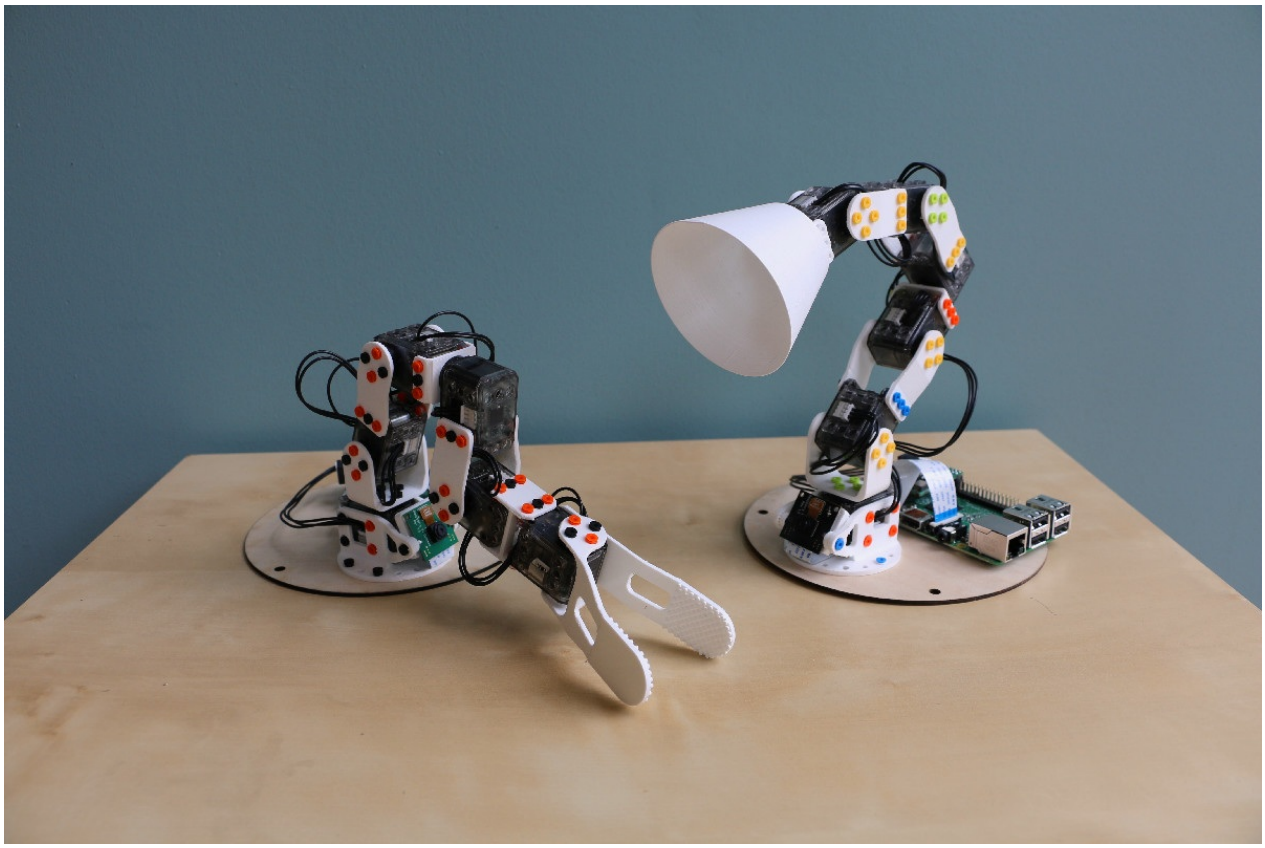
They can be used as it is, or hacked to explore new shapes, add sensors, etc...

To get your own Poppy robot, you can either:

- Get all the parts yourself by following the bill of materials (see below).
- Buy a full Poppy robotic kit from our official retailer.

Poppy Ergo Jr

The Poppy Ergo Jr robot is a small and low cost 6-degrees-of-freedom robot arm. It is made of 6 cheap motors (XL-320 Dynamixel servos) with simple 3D-printed parts.

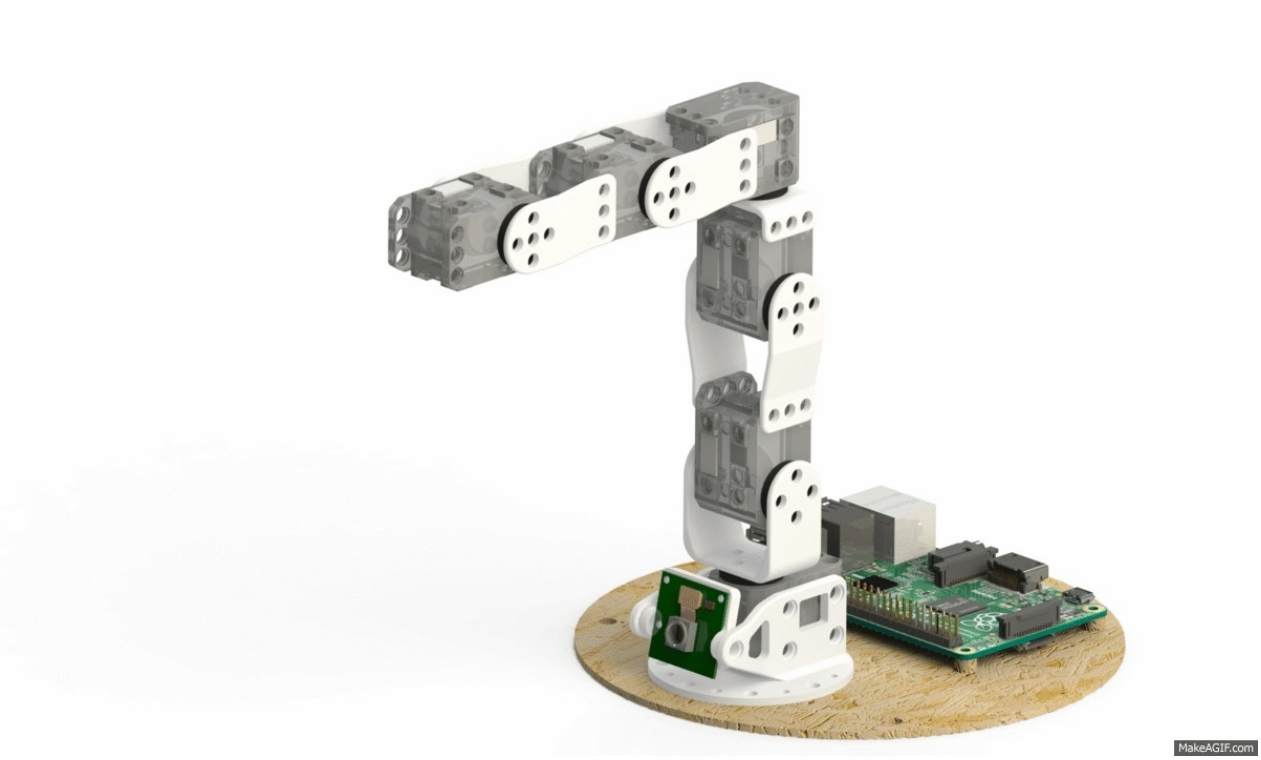


The 3D parts were made so they can be easily printed on a basic 3D printer. The motors are only 20€ each. Its electronic card access is simple. It makes it easy to connect extra sensors and is well suited for pedagogical purposes.

You can choose among three tools at the end of its arm:

- A lamp.
- A gripper.
- A pen holder.

The rivets used make it easy and quick to change the tools. You can adapt it depending on the type of activities you are doing.



The Ergo Poppy Jr is ideal to start manipulating robots and learn robotic without difficulties. It is simple to assemble, easy to control and low price.

You can get all the parts yourself following the bill of materials (BOM) and print the 3D parts available in the STL format.

For more information, check the assembly guide of the Ergo Jr.

Poppy Humanoid

It is a 25-degrees of freedom humanoid robot with a fully actuated vertebral column. It is used for education, research (walk, human-robot interaction) or art (dance, performances). From a single arm to the complete humanoid, this platform is actively used in labs, engineering schools, FabLabs, and artistic projects.

You can get all the parts yourself following the bill of materials (BOM) and print the 3D parts available as STL, STEP and Solidworks 2014 format.

Poppy Torso

It is the upper part of Poppy Humanoid (13 degrees of freedom). Poppy Torso is thus more affordable than the complete Poppy Humanoid. It makes it a more suitable solution for educational, associative and makers contexts. Poppy Torso can be a good medium to learn science, technology, engineering and mathematics (STEM).

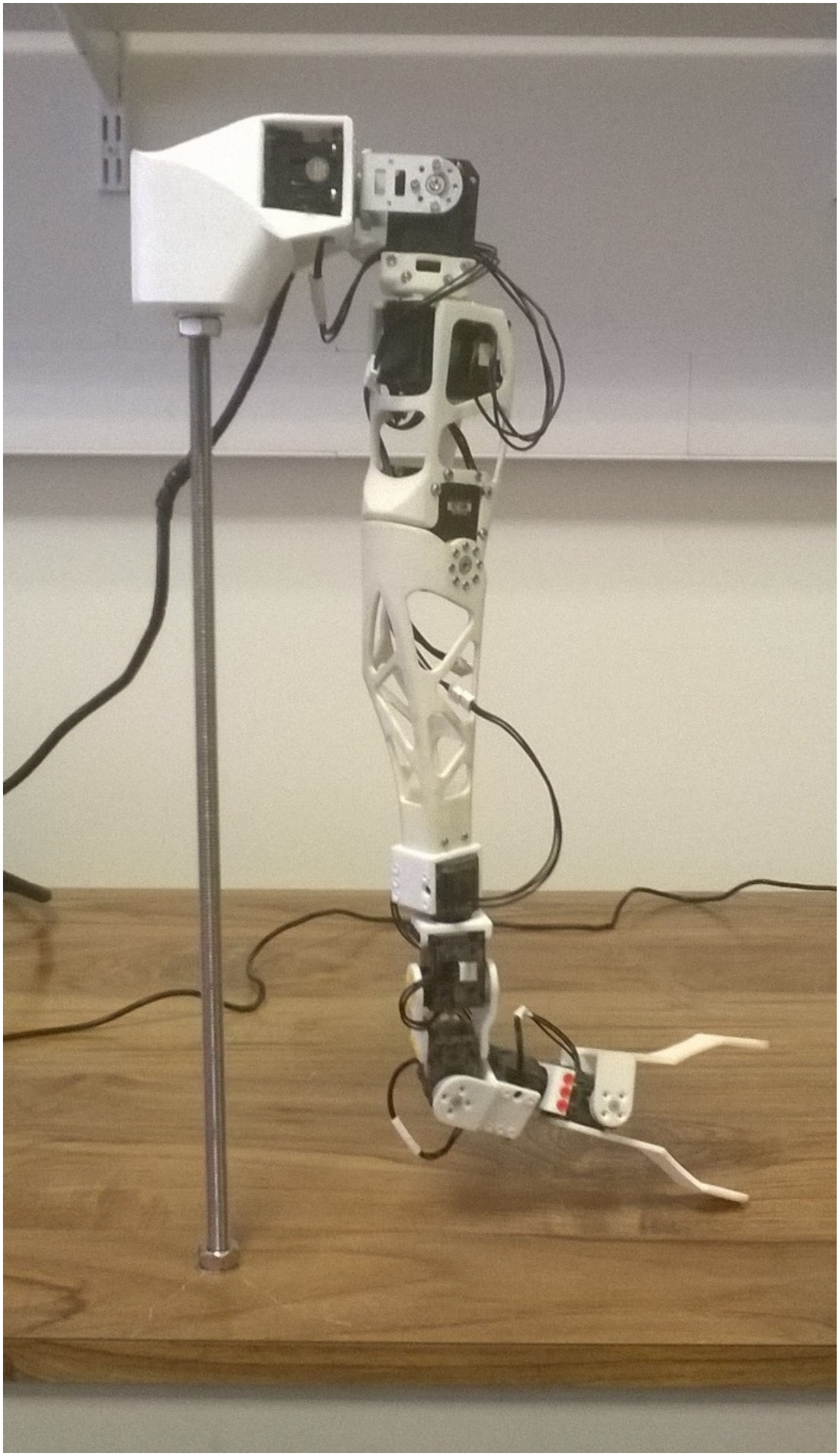
You can get all the parts yourself following the bill of material. The 3D models for the parts are the same as Poppy Humanoid, without the legs and with an extra suction cup support.

Other interesting Poppy Creatures

A key aspect of the Poppy Project is to stimulate creativity and experimentation around robotics. We try to provide all the tools needed to design new robots based on the same technological bricks. Some new creatures are in development within the community. Some of them are presented in the section below.

Poppy right arm (work in progress)

Poppy right arm is a Poppy creature based on a right arm of Poppy Humanoid, with 3 additional XL-320 motors at the end to improve the reach and agility of the arm. It used the same gripper tool used in the Ergo Jr, designed to grab simple objects.



The project was realized during an internship at Inria Flowers by Joel Ortiz Sosa. Find more info and the sources in the repository.

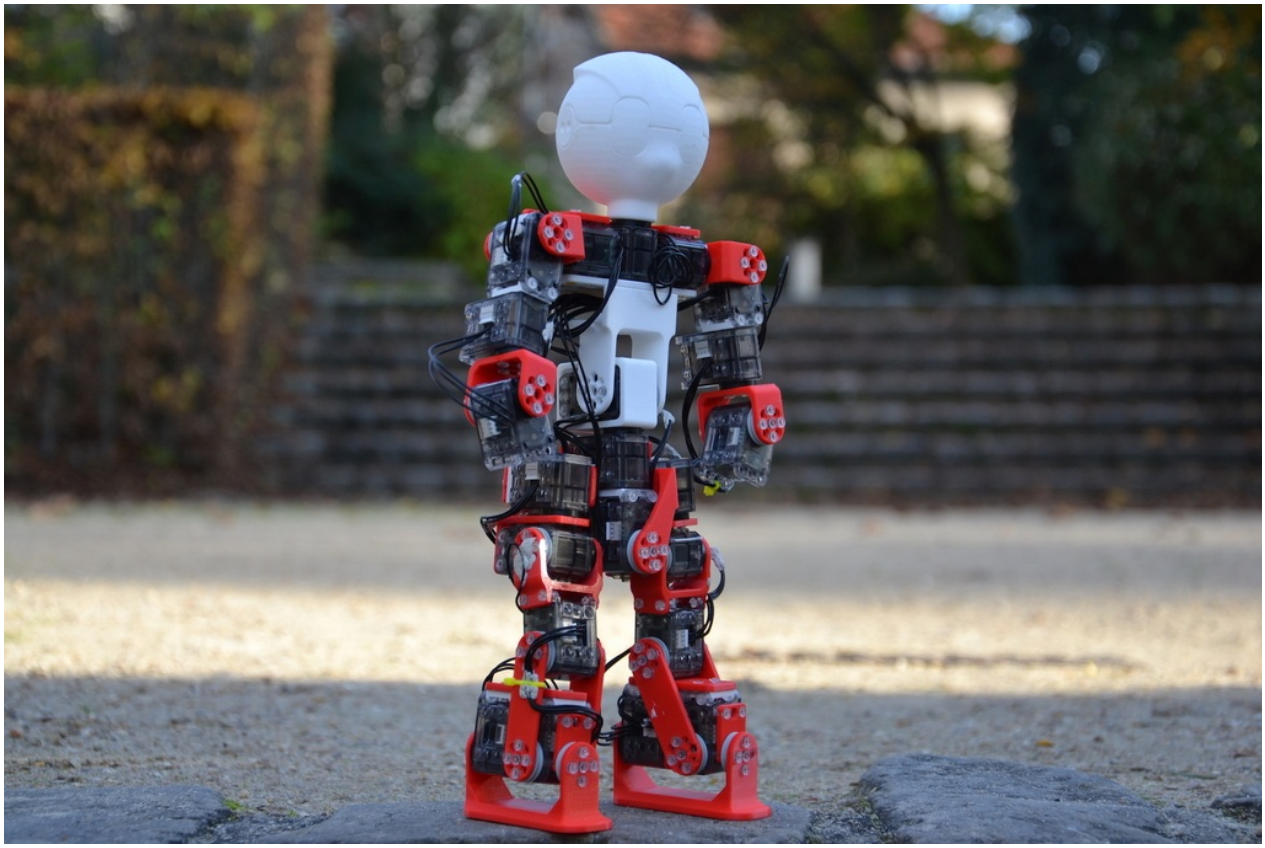
Small and low-cost humanoids

Heol

Heol - meaning "sun" in Breton - is a 34cm tall humanoid robot made by the association Heol robotics. 23 motors compose it, all other parts are 3D printed. It also uses the pypot library for its movements.

Heol's purpose is to put a smile on ill children's faces. It can be an educational tool by becoming a learning support materials for programming and mechanical design.

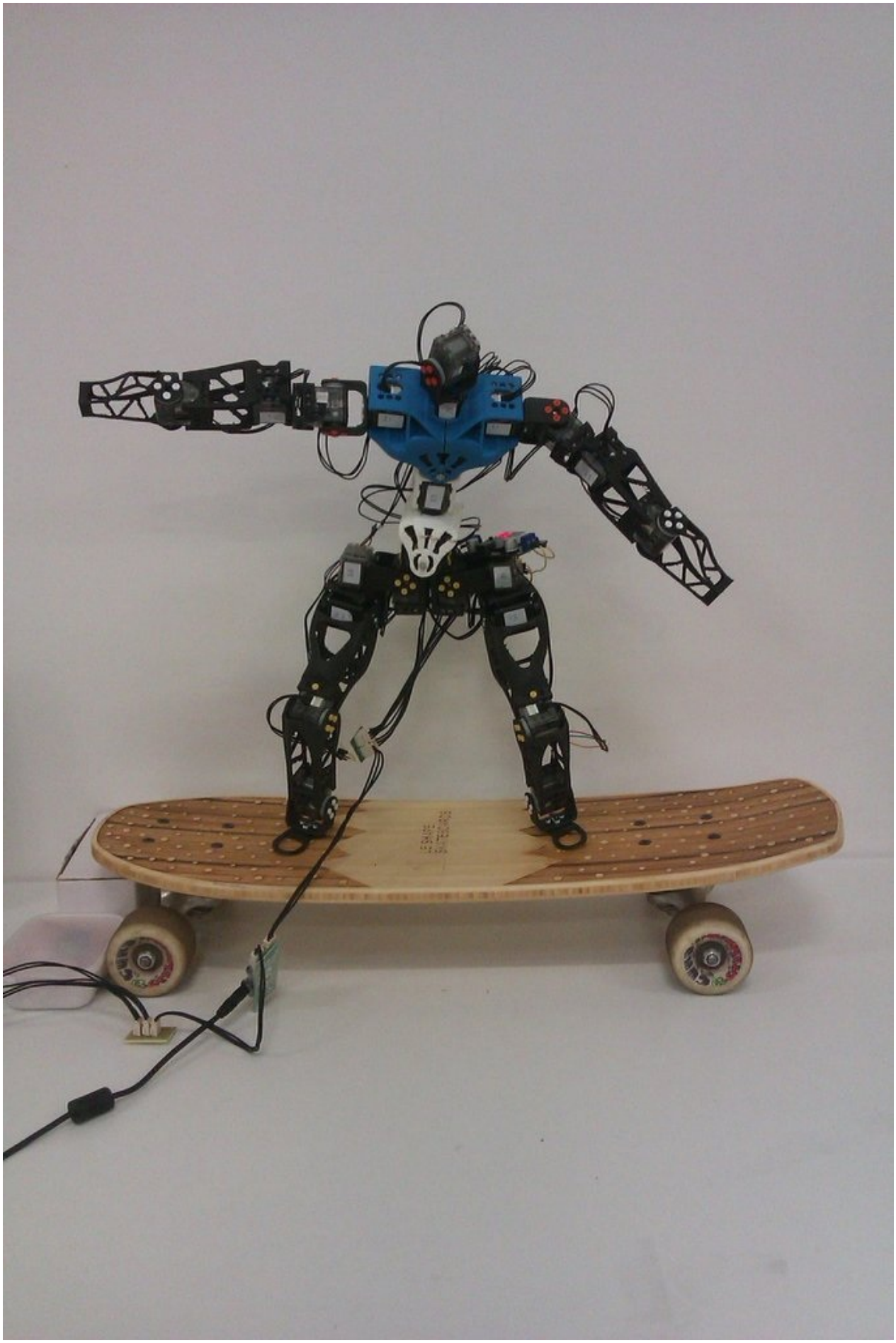
Its participation in the RoboCup (World Cup football for robot) is also envisaged.



Poppyrate

It a robot based on Poppy Humnaoid. It aims at developing a more affordable version thanks to its smaller size and the use of cheaper motors. The size reduction also makes it easier to print the parts on a standard 3D-printer. Goals also involved making it as mobile and customisable as possible while maintaining compability with Poppy software.

Poppyrate will be sold as a kit (with and without the 3D parts) It has been designed by the socity ZeCloud.



For more information, check their Website - Twitter - Facebook!

Build the robot

Depending on the Poppy robots you are planning to use, the assembly time, required skills, tools and difficulty may vary a lot. Building an Ergo Jr should take about one hour and no specific tool is needed while assembling an entire Poppy Humanoid may take a few days and quite a lot of screws!

This section intends to give you hints and a glimpse of some critical points so you aware of them before digging into the construction. We will also point to each dedicated chapter where you will find the resources and detailed step by step assembly procedure for each robot.

Assembling an Ergo Jr

> You can find a full assembly documentation in the chapter step by step assembly of an Ergo Jr.

The Ergo Jr robot was designed to be a simple little robot, cheap and easy to use. The 3D parts were made so they can be easily printed on a basic 3D printer and the motors (6 XL-320 Dynamixel servos) are only 20$ each.

The Ergo Jr is very easy to build and its end effector can be easily changed - you can choose among several tools: a lamp, a gripper, a pen holder...

Thanks to OLLO rivets the robot is very simple to assemble. These rivets can be removed and added very quickly with the OLLO tool. It should not take more than one hour to entirely built it, which allows great design freedom.

Except from **checking the motor orientation**, there is not really any pitfall. If you are familiar with Lego bricks, you should be able to assemble an Ergo Jr without much problem! Rivets were made to be as easy to assemble than to disassemble, so in case of problem you can just start over!

Also make sure, to **configure your motors** before assembling the robot as it is harder to do after!

Assembling a Torso or a Humanoid

> You can find a full assembly documentation in the chapter step by step assembly of a Poppy Humanoid.

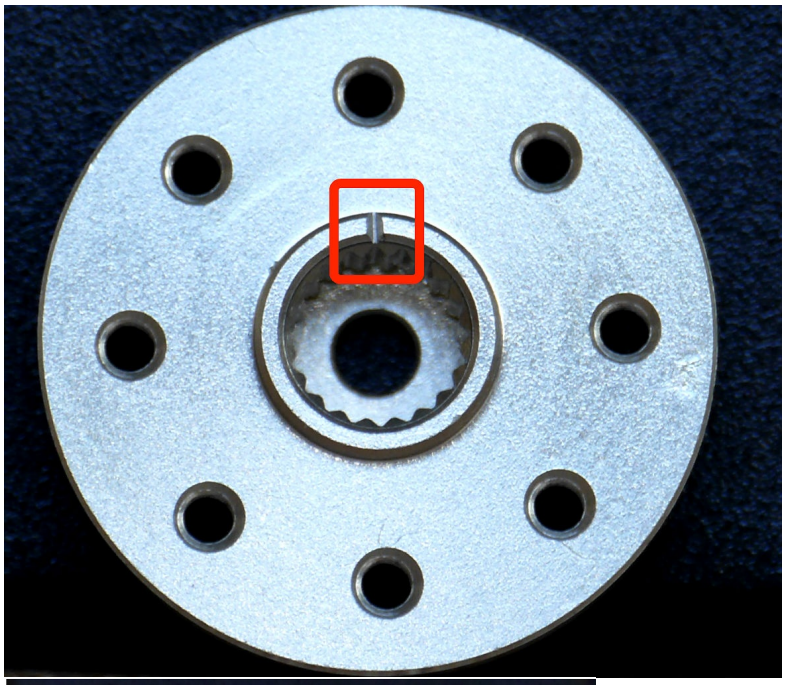
Building a Poppy Torso or a Humanoid is more complex than a Ergo Jr but it is not really more complicated than building a Meccano or some Swedish furniture. It mainly consists on those few steps:

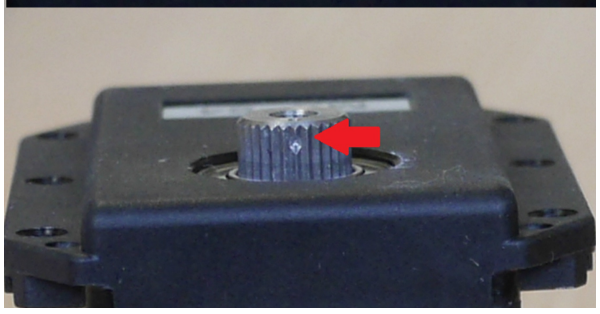
- assemble the horn on each motors: **you will have to be really cautious about the motor zero position!**
- configure the motors so they match the *poppy configuration*
- use a lot of screws to connect all 3D printed parts to the motors
- do a bit of electronic for the embedded board inside the head: this can be a bit tricky if you are not familiar with electronics.

Patience and precision are your allies, but in case of errors do not panic: Poppy is a robot intended to be assembled and disassembled. If you pay attention to the few **warnings** bellows, and with a few trials and errors you will have a working Poppy Torso or Poppy Humanoid:

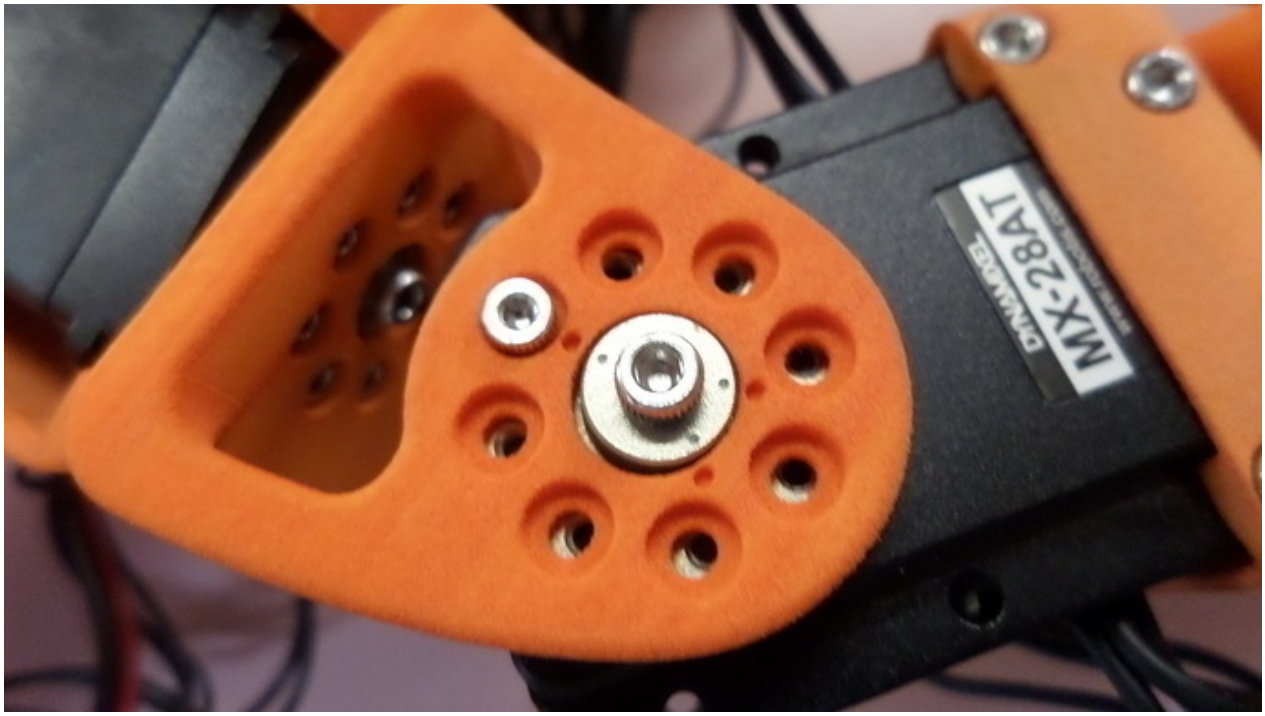
Warning 1: The Poppy humanoid and torso robots are built using mainly MX-28 and MX-64 Dynamixel servomotors. They are pretty powerful and may be harmful to your fingers or materials. So be very careful and put the robot in a free space while testing it!

Warning 2: Put the dot on the horn at the same point than the dot on the servo axis.





Warning 3: Adjusts the three dots of the motors with the three dots of the structural part.



Warning 4: Use thread locker to prevent vibrations from untying the screws. However, dipping the extremity of the screw on the thread locker is enough (a drop for each screw hole is too much). Otherwise disassembling your robot can be very hard!

Step by step guide for the asssembly :

- Guide for the Humanoid
- Guide for the Torso

Start and connect the robot

In this section, we will describe how to start your robot and give an overview of the possibilities to access it.

Setup the software

Poppy creatures come with an embedded board which job is to control motors and access the sensors. For simplicity purpose, this computer can be remotely access through a web interface. It makes it easy to control the robot from your own computer or tablet without having to download/install anything.

There are two ways to setup the board for your Poppy robot:

- **the easy way**: use a pre-made ISO image of the Poppy operating system and write it to an the SD-card
- **the hard way for advanced users**: install everything from scratch

> If you are planning to use a simulated robot, you must install the software on your personal computer. Follow the instructions for setting up the simulation.

Easy and recommended way: use the Poppy SD-card

The easiest and quickest way - by far - is to use an already made system image for a SD-card. ISO images come with everything pre-installed for your Poppy robot. It is also a good way to ensure that you are using exactly the same software as we are. Thus, you will avoid most problems.

> Poppy robotic kits come with a ready to use SD-card. So, you do not have anything special to do.

The images can be found in the release of each creatures:

- for the Poppy ErgoJr
- for the Poppy Torso
- for the Poppy Humanoid

They can be written to a SD-card (at least 8 Go) by using classical utility tools. Once the SD-card is ready, just insert it into the board. Then when you switch on your robot it should automatically start and you should be able to connect to its web interface.

> More details can be found in the startup section.

Advanced way: DIY, install everything from scratch

The advanced way mainly consists in installing everything needed from scratch. This follows the same procedure as we use to generate the image for SD-cards. We mention this possibilty here as it can be useful if:

- You are **working with a simulated robot** and thus have to manually install all the required software on your computer, this procedure could be a good place to see how this can be done on a Raspberry-Pi and adapted to another computer,
- you want to customize the environment,
- or simply if you like to understand how it works.

> We try to keep this installation procedure as generic as possible. Yet, some details may vary depending on your operating system or your computer. Moreover, the installation from scratch required some good knowledge of how to install and setup a python environment.

Depending on what you want to do all steps are not necessary required. In particular, if you want to control a simulated robot, you may just want to install the python libraries for Poppy.

> More details can be found in the Installation for advanced users section.

Setup the network

Once your Poppy is assembled and its software is ready, the next step is to connect it to a network. The goal is to let you remotely access the robot from your computer or smartphone/tablet, control and program it.

They are two main ways to connect your robot to your computer/tablet/smartphone:

- Connect both the robot and the computer to the same network (e.g. the box of your home or the school network).
- Directly connect your robot to your computer using an ethernet cable.

> While directly plugging the robot to a computer works for most users. It seems that in some strange cases it refuses to work.

To find the address of your robot on the network, we use the standard Zeroconf protocol. It allows you to use the robot hostname: *"poppy.local"* as its address. This should works without any configuration under Mac OS and GNU/Linux. But it required to install *Bonjour Print Services* on Windows. If you prefer, you can use the IP address assigned to your robot instead. If you are not administrator of your network this can be a tricky information to find. In this case the first procedure should be preferred.

To check that everything is setup correctly, you can go to the following url using your favorite web browser: http://poppy.local/. You can replace *poppy.local* by the IP address of your robot (something similar as http://192.168.0.42).

> If you are not familiar with network configuration or have no idea what the previous paragraph poorly tried to explain, you should see with the IT network engineer, how this can be done.

Use the web interface

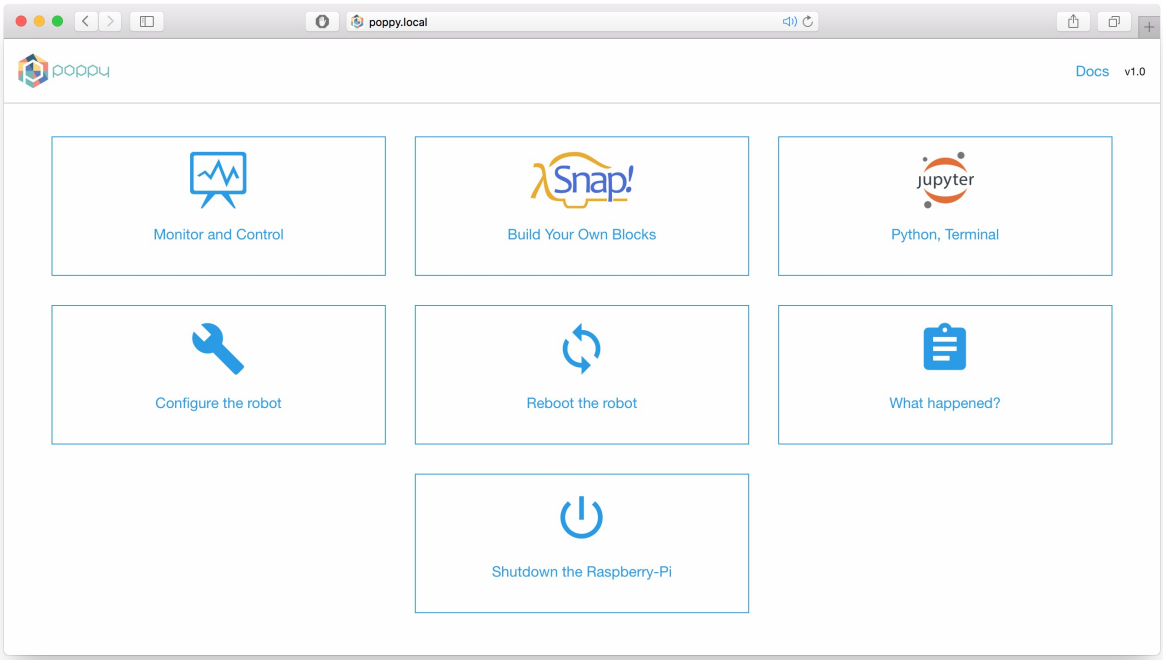
The web interface is the central point to control, program and configure your robot. It can be used to:

- Monitor and Control the robot
- Program it in Snap!
- Program it in Python
- Configure the robot (change its name, enable/disable the camera, update)
- Reset and shutdown the robot

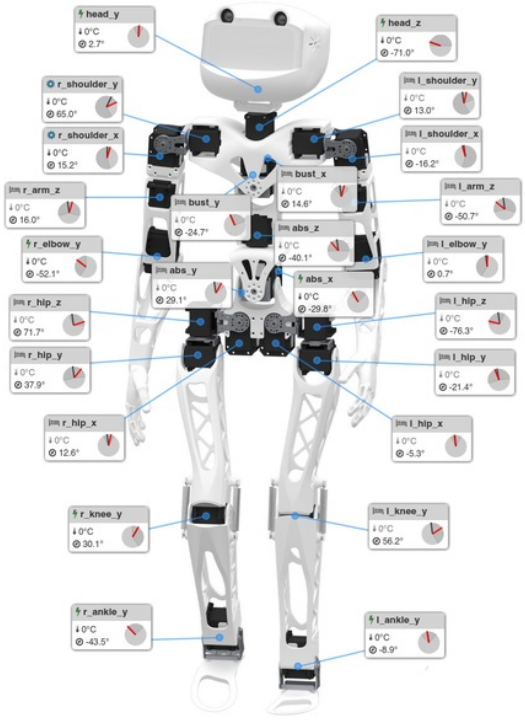
To access this interface, you simply have to go to the URL using your favorite web browser:

- http://poppy.local (if you changed the name of your robot, simply replace *poppy* by its new name)
- or using directly its IP address

You should see something like:

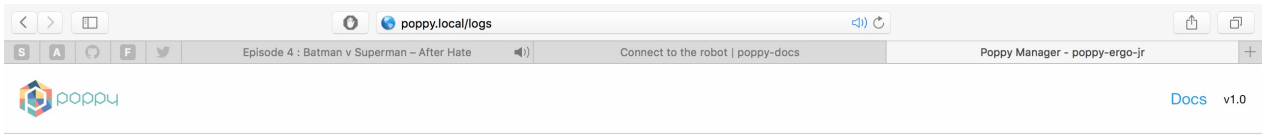


The buttons can be used to navigate to the different features. For instance, if you click on the *Monitor and Control*, you will access the monitor web app:



This let you turn on/off the motors of your robot, monitor them, and start/stop behaviors.

The *What happened* button is where you should look for more information is something goes wrong. Here is a screenshot of what you should see if everything goes well:



Logs

```
Attempt 1 to start the robot...
SnapRobotServer is now running on: http://0.0.0.0:6969

You can open Snap! interface with loaded blocks at "http://snap.berkeley.edu/snapsource/snap.html#open:http://10.204.4.89

HTTPRobotServer is now running on: http://0.0.0.0:8080

Robot created and running!
```

Program the robot

Poppy robots are designed to be easily programmed. They are three main options presented here:

- using Snap!, a variant of Scratch the visual programming language,
- using Python and leveraging the power of the whole API,
- through the REST API which let you interface Poppy robots with other devices or any programming language.

> As for the rest of the project, all our libraries are open source and available on GitHub.

Using Snap*!*

Snap*!* is a visual programming language - a variant of the very famous Scratch language. It is a block based drag-and-drop programming language that allows for a thorough introduction of IT. It runs in your browser as it is implemented in JavaScript. You do not need to install anything to start using it. It is open sourced and actively maintained.

We developed a set of custom blocks for Poppy robots that let you send motor commands and read values from the sensors of your robot. This let you to directly jump into controlling and programming your without any syntax/compilation issue. Thanks to Snap*!* live interaction loop you simply have to click on a block to send its associated command to the robot. Snap*!* also naturally scales to more complex projects.

A dedicated chapter will guide you in what you can do with Snap*!* and Poppy robots.

Using Python

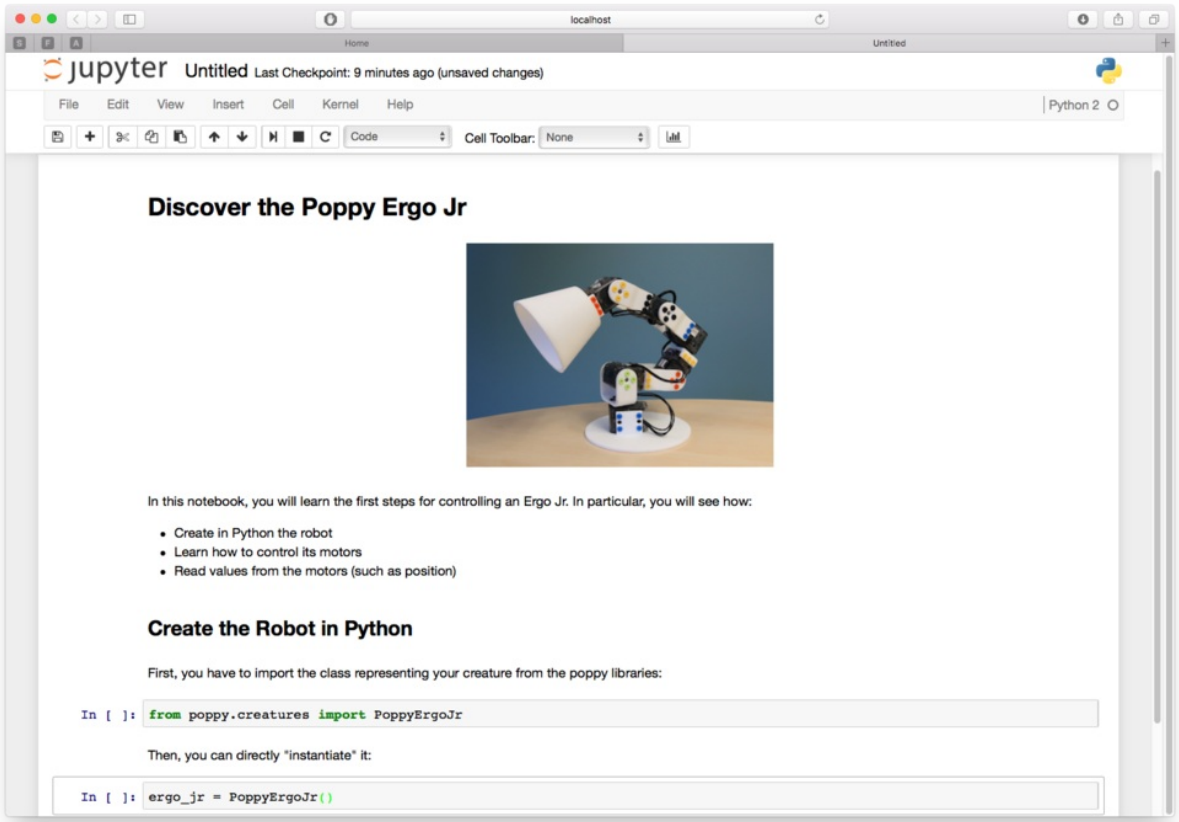
Poppy libraries have been written in Python, to allow for fast development and extensibility and to benefit from all existing scientific libraries. Python is also a well-known language and widely used in education or artistic fields. By programming Poppy in Python, you will have access from the very low-level API to the higher levels.

The API has been designed to allow for very fast prototyping. Creating a robot and starting to move motors should not take more than a few lines:

```
from poppy.creatures import PoppyErgoJr

jr = PoppyErgoJr()
jr.m3.goal_position = 30
```

We are also big fan of the Jupyter Project and notebooks. Notebooks are documents which contain both Python code and rich text elements like equations, pictures, videos. They can be edited from the Jupyter Web interface which allow users to program Poppy robots directly from a website hosted on the robot computer. We think that this is a powerful tool permitting the creation and sharing of live code, results visualizations and explanatory text combined in a single document.

Most of the tutorials, experiments or pedagogical activities that we and the community develop are available as notebooks.



An updated gallery of notebooks can be found here. Contribtions welcomed!

Through the REST API

On top of the Snap*!* and Python options, we wanted to provide another way of accessing and controlling your robot from any device or language. Poppy robots are providing a REST API. The most important features of the robot can be access through HTTP GET/POST requests.

From a more practical point of view, this allows you to:

- **Write bridges to control Poppy robot in any language** (awesome contributors have already written Matlab and Ruby wrappers).
- **Design web apps** connected to your robot, such as the monitor interface (also a contribution!).
- Make your **robot interact with other connected devices** such as a smartphone, intelligent sensors, or even your twitter account...

The REST API is still a work in progress, will change and is clearly ill documented! For more information you can have a look here our on the forum. A well designed, stable and well documented REST API is expected for the next major software release.

Visualize the robot in a simulator

Simulated Poppy Creatures

Simulated versions of all Poppy robots (Humanoid, Torso, and Ergo Jr) are available.

Connection with two main "simulators" were developed:

- using V-REP: a virtual robot experimentation platform
- using a 3D web viewer: lighter but without physics support

> At the moment, only the Poppy Ergo Jr can be used in the web visualizer. If you want to simulate other creatures, you should use V-REP. Support for the other robots is planned but not expected in the near future.

We think simulation can be a powerful tool. It allows the development and test of programs without the need of having a real robot. This is especially useful:

- To discover and try the robot possibilities without having to spend real money.
- In a context where multiple users share a robot. For instance in a classroom where each group can work using the simulator and validate their program on a real robot.
- To design and run complex and time consuming experiments.

We try to make the **switch from a simulated Poppy robot to the real one as transparent and as simple as possible**. Most of the programming documentation is actually valid for both simulated and real robots. The chapter *From simulation to real robot* will guide you in the few steps to transform your program running in simulation to one working with a real robot.

> If you want to use Poppy robots using a simulator you will have to install some of the poppy libraries locally on your computer.

Install the needed software

While the physical robots come with libraries pre-installed, they are not integrated in the simulators. Thus, you need to install them on your computer. More details of what you will have to do is given in the section below. You will also not have access to the robot web interface. You will have to manually launch the different services to start programming your robot (the Jupyter server for Python notebooks, or the Snap*!* server).

> We are hoping to have a one-click app for Windows/Mac/Linux with everything setup at some point. Yet, this is not expected to be available in the near future.

To start controlling a simulated Poppy robots, either using V-REP or the web visualizer, you will need:

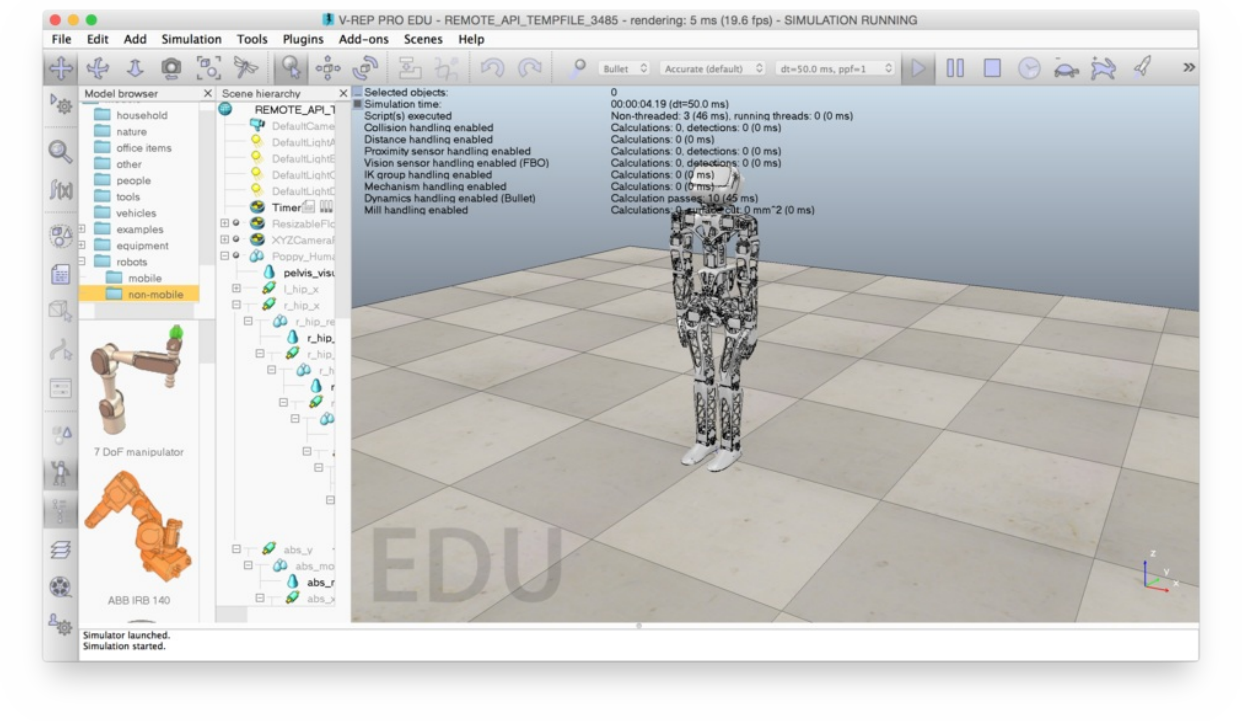
- To have a working Python, we strongly recommend to use the Anaconda Python distribution. It works with any version >=2.7 or >=3.4. Prefer Python 2.7 if you can, as it is the version we used.
- To install the Poppy libraries: pypot and the library corresponding to your creature (e.g. poppy-ergo-jr).

> Details about those steps can be found in section *Install everything locally for using a simulator*.

Using V-REP

V-REP is a well known and powerful robot simulator. It is widely used for research and educational purposes. Moreover, it is available for free under an educational license. It can be download from this website (works under Mac OS, Windows and GNU/Linux).

> It is important to note that as V-REP is simulating the whole physics and rendering of the robot, it may be slow if you do not have a powerful computer (especially the GPU card).

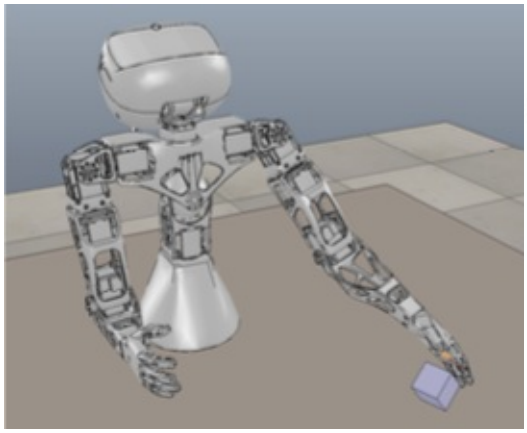


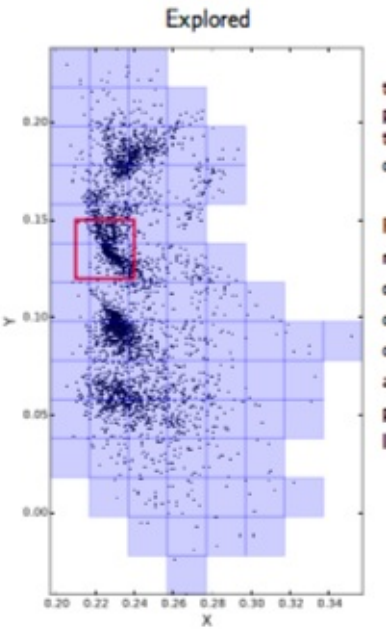
All main Poppy robots are available in V-REP:

- Poppy Humanoid
- Poppy Torso
- Poppy Ergo Jr

V-REP can be used to learn how to control motors, get information from sensors but also to interact with the simulated environment. It can be controlled using Python, Snap*!* or through the REST API. Here, are some examples of what the community has already been doing with it:

- A pedagogical activity to discover the different motor of your robot and how they can be controlled.
- A scientific experiment, where a Poppy Torso is learning how to push a cube on a table in front of it

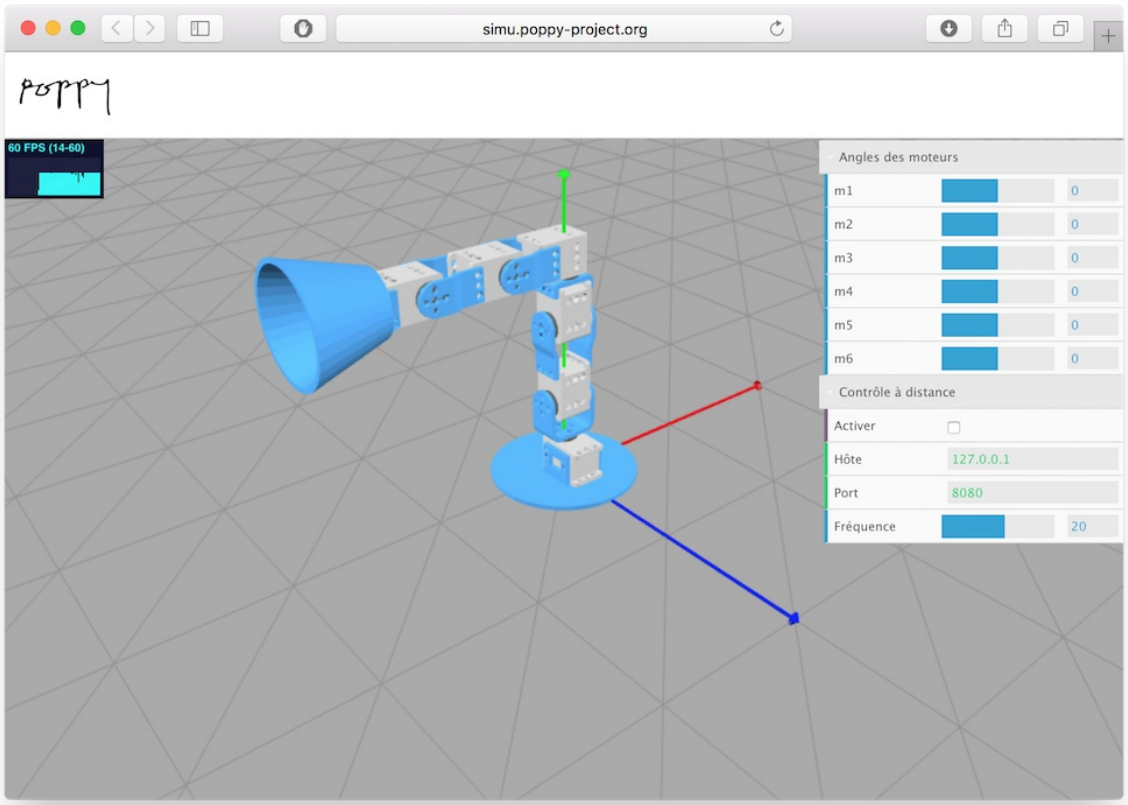




> Even if we try, to reproduce the robot behaviour and functioning, some differences remain. In particular, if you make a robot walk in simulation that does not necessarily mean that it will walk in the real world (and vice-versa).

Using our web visualizer

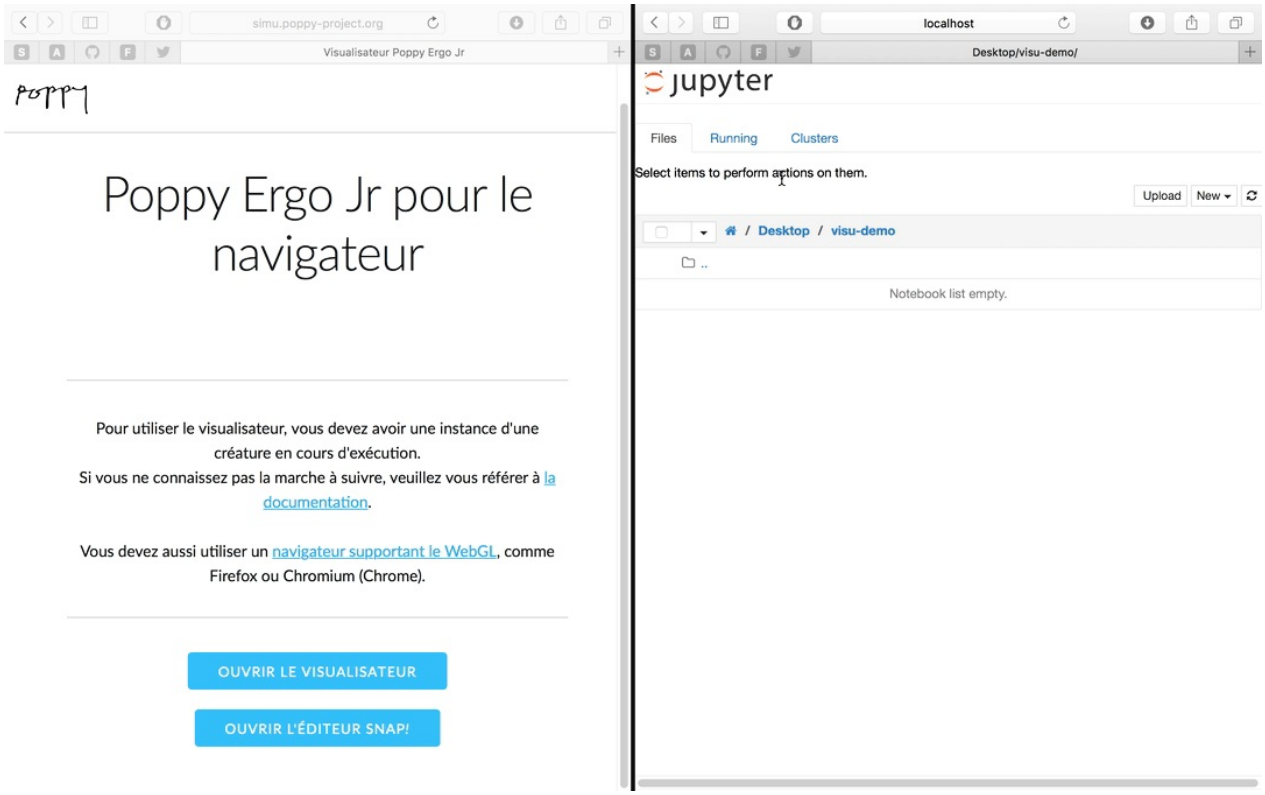
Our web visualizer - based on the Three.js library - will show you a 3D representation of a Poppy robot. For this, you will need to connect it to either a real robot (through the REST-API) or to a simple mockup robot running on your computer. You simply have to set the host variable from within the web interface to match the address of your robot.



> A mockup robot can be started via the poppy-services command. For instance: `poppy-services --poppy-simu --snap poppy-ergo-jr`

As for V-REP, you can control your Robot using Python, Snap*!*, or the REST API. Yet, there is no physics simulation so its lighter but you will not be able to interact with objects.

Here is an example with Python:


simu.poppy-project.org
Visualisateur Poppy Ergo Jr
Poppy Ergo Jr pour le navigateur
Pour utiliser le visualisateur, vous devez avoir une instance d'une créature en cours d'exécution.
Si vous ne connaissez pas la marche à suivre, veuillez vous référer à la documentation.
Vous devez aussi utiliser un navigateur supportant le WebGL, comme Firefox ou Chromium (Chrome).
OUVRIR LE VISUALISATEUR
OUVRIR L'ÉDITEUR SNAP!
localhost
Desktop/visu-demo/
jupyter
Files
Running
Clusters
Select items to perform actions on them.
Upload
New
/ Desktop / visu-demo
..
Notebook list empty.

Overview of projects developed by the community

The Poppy community gathers an interdisciplinary community of beginners and experts, scientists, educators, developers and artists. Various and interdisciplinary robotic creations have emerged. Some of them are illustrated below.

School of Moon

The School of moon play has been created by the contemporary dance company Shonen lead by the choregrapher Eric Minh Cuong Castaing.

The stage is shared by children, two dancers (Gaëtan Brun Picard and Ana Pi), 3 Nao robots and two Poppy Humanoid robots. This play is a metaphor of the creation of a post-humanity in three acts: the Man, the Man and the machine, and the machine.

The representations are localized, meaning that the children dancing come from the local town. They also are specific sequences depending on the robots existing in the city.

The artistic challenges are:

- Directing children on stage
- Having interaction between humans and robots on stage
- Having robots on stage

The creation was focused on 4 time periods:

- 2 weeks of residence in September 2015 in CDC of Toulouse (France)
- 2 weeks of residence in KLAP of Marseille (France) in December 2015
- 4 weeks of residence in *Ballet National de Marseille* (France) in January 2016
- 2 weeks of residence in Düsseldorf (Germany)

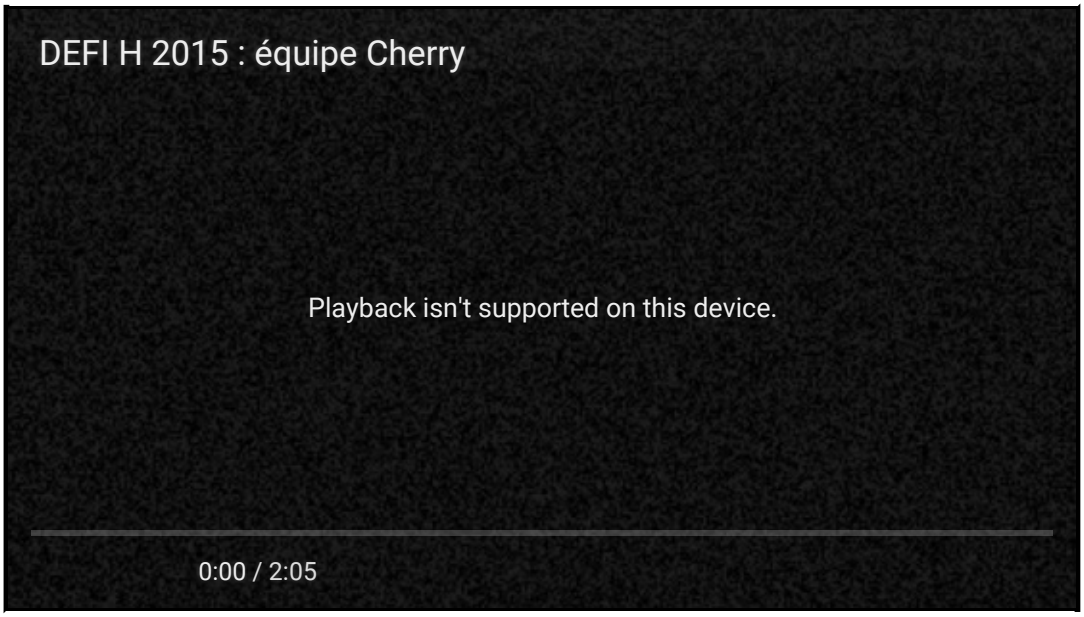
Cherry Project

The Cherry project is a community project to develop scenarios regarding breaking isolation of children at the hospital.

This project uses Poppy robot as a companion for hospitalized children in primary school. Cherry can compensate a social rupture during hospitalization. It mediates between the child, his friends, his family and teachers and can talk to him or play some games.

It also acts on a pedagogical level, to encourage the child to interact with the school, by offering quizzes and educational games.

One last prospect is to assist the hospital staff in the therapeutic education. Indeed, sometimes a message is more acceptable by the child if it is issued by the robot rather than an adult dressed as a physician.

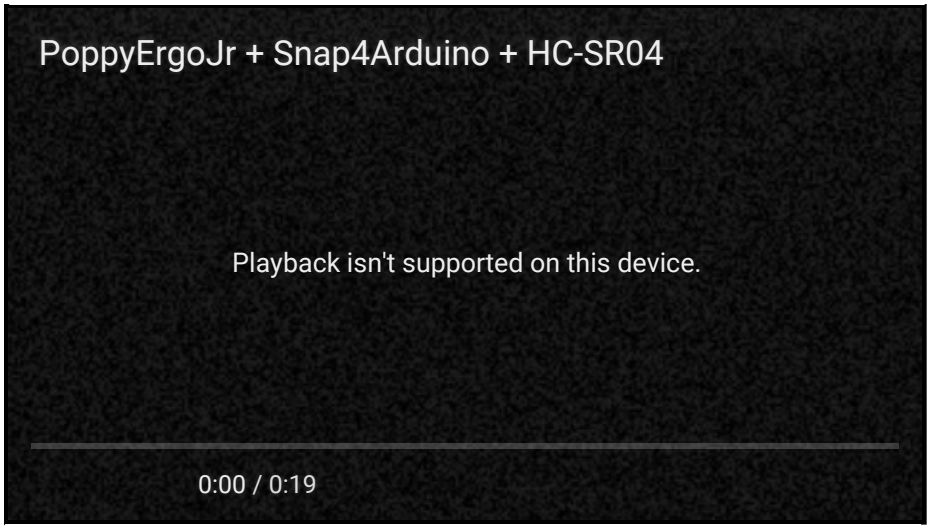


More information:

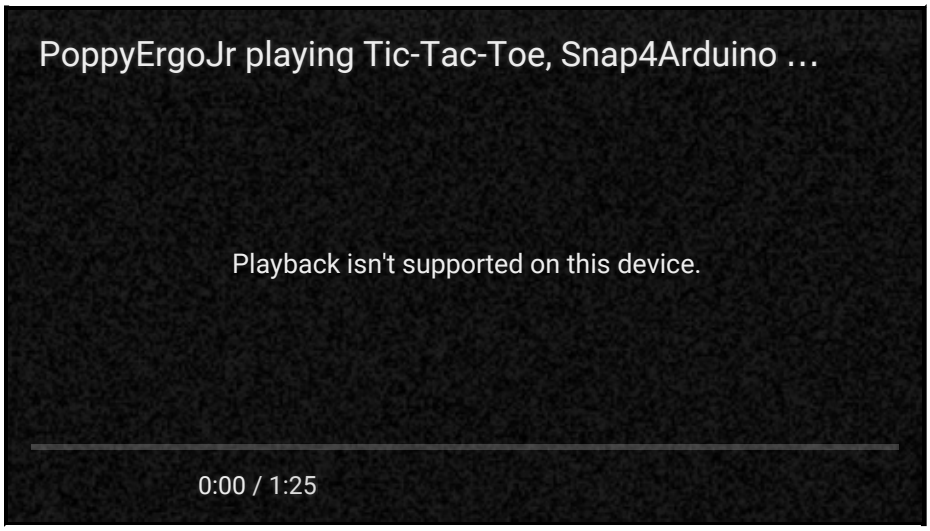
- The facebook page
- The twitter
- The wordpress
- The github (with a short wiki)

Connect Poppy and Arduino thanks to Snap4Arduino

Gilles, teacher and maker at night, has developed many projects based on the Poppy Ergo Jr and Arduino. For connecting both worlds, it uses Snap4arduino. Then, it becomes really easy and elegant to make them communicate. You can simply mix Arduino with Poppy blocks and tadam you can control your robot with any Arduino based sensor.



Then the only limitation is your creativity! For instance, ou can make Poppy Ergo Jr plays TicTacToe:



The detailed documentation can be found in the section Control Poppy with Arduino.

Installation

Poppy project is vast, there are a lot of installation paths depending on what you want to do:

- You want to install a tangible Poppy robot
- You want to try Poppy robots in a simulator or in a web viewer
- You want to do some advanced stuff with a tangible robot

> Whatever you want to do, all section below suppose that you have a zeroconf client (also called *Bonjour*) installed on your computer. It is not mandatory but otherwise, you will be on your robot IP address.

You want to install a tangible Poppy robot

Poppy creatures are controlled by a small embedded computer: a Raspberry Pi or an Odroid board. The operating system of this computer is hosted on a SD card (you can also use an MMC for the Odroid).

You have two possible states:

- You already have a pre-formated SD card with the Poppy operating system (provided by one of the Poppy distributors). You have nothing to install, you are ready to go to the assembly section.
- You have an empty SD-card, **so you have to download and write the operating system on the SD card.** This is the most common case.

If you are a Linux rocky and want to try yourself our unstable install scripts, you can go the *install a poppy board* chapter.

You want to try Poppy robots in a simulator or in a web viewer

- Install Poppy softwares on your computer
- Install V-REP simulator

You want to do some advanced stuff with a tangible robot

If you want to install yourself the system of the Poppy robots with our unstable install script:

- Install a Poppy board

If you want to control a tangible robot from your personal computer, you have to:

- Install Poppy softwares on your computer
- Check USB to serial drivers if you are on Windows

Zeroconf

Zeroconf also called Bonjour (name of Apple implementation) is set of technologies that allow more easily communication between computers without configuration.

> Zeroconf is not mandatory on your computer to use Poppy robots, but we will assume it is installed. It is more convenient and readable for the documentation.

Local domain name

Zeroconf client *publishes* a decentralised local domain name (mDNS) with the '.local' top level domain. It means that you can join any local local computer by its hostname with the '.local' suffix instead of its IP address.

With a zeroconf client, to `ping` a computer called (hostname) 'goldstine', you can simply do:

```
$ ping goldstine.local
64 bytes from 192.168.1.42: icmp_seq=0 ttl=54 time=3.14 ms
[...]
```

You no longer need to look for its IP address on your local network; you don't even need to understand what an IP address is.

It also work on your web browser. To open the website hosted on the robot computer called 'goldstine', you have to open: http://goldstine.local on your favorite web browser URL field.

Link-local IPv4 addresses

Among other Zeroconf tools, there is an implementation of decentralized DHCP (IPv4LL), which allow computers obtain an IP and connect each others **without** a DHCP server.

The auto-adressed IP is in the APIPA range, from 169.254.0.0 to 169.254.255.255.

You can plug a robot to your computer **directly** on your computer with an Ethernet cable, **without** any router and connect it with its local domain name (hostname.local).

> You will be able to use the local-link IPv4 address **only** if you installed your robots after end of May 2016. Previously *avahi-autoipd* packet was missing.

Installation

- On Windows, you have to install *Bonjour print services for Windows* (yes, it is an Apple software).
- On GNU/Linux, you have to install *avahi-daemon* (mDNS) and *avahi-autoipd* (IPv4LL), it may or may not be installed by default depending on your installation. Run `sudo apt-get install avahi-daemon avahi-autoipd` on Debian/Ubuntu or `sudo yum install avahi-daemon avahi-autoipd` on Fedora.
- On Mac OSX it works out of the box.

You ready to follow your installation path.

Alternatives to find the IP address of a computer on your local network

If you cannot (or doesn't want to) install a zeroconf client on your personal computer, you can use one of the following methods to find the IP address of your robot.

- You can use Fing, famous for its Android and iOS applications,
- Nmap (only GNU/Linux and MAC OSX) if you are not afraid of command line interfaces.
- You can also go to your router web interface (with its IP address on your web browser like http://192.168.0.1 or http://192.168.1.1 or http://192.168.0.254 or http://192.168.1.254), you should have a section of connected hosts.

Startup with a Poppy robot

> This chapter is only for people who want to control a tangible robot. If you intend to control a simulated robot on your computer, look at the simulation install path.

Poppy creatures are controlled by a small embedded computer: a Raspberry Pi or an Odroid board.

The operating system of this computer is hosted on a SD card (you can also use an MMC for the Odroid).

You may be in two kind of cases:

- You already have a SD card with the Poppy operating system (provided by one of the Poppy distributors for example). You're ready to go to the assembly section.
- You have an empty SD-card, so you have to download and write the operating system on the SD card.

> The Poppy creatures operating system use a GNU/Linux distribution, but you won't have to any knowledges on Linux to install the image on the Raspberry Pi. You will only need a computer with a SD card reader/writer to write the image on the SD card.

Download the image of the operating system

You have to choose the image (file in *.img.zip) to download depending on your Poppy creature and the targeted board:

- Ergo Jr
- Poppy Torso
- Poppy Humanoid

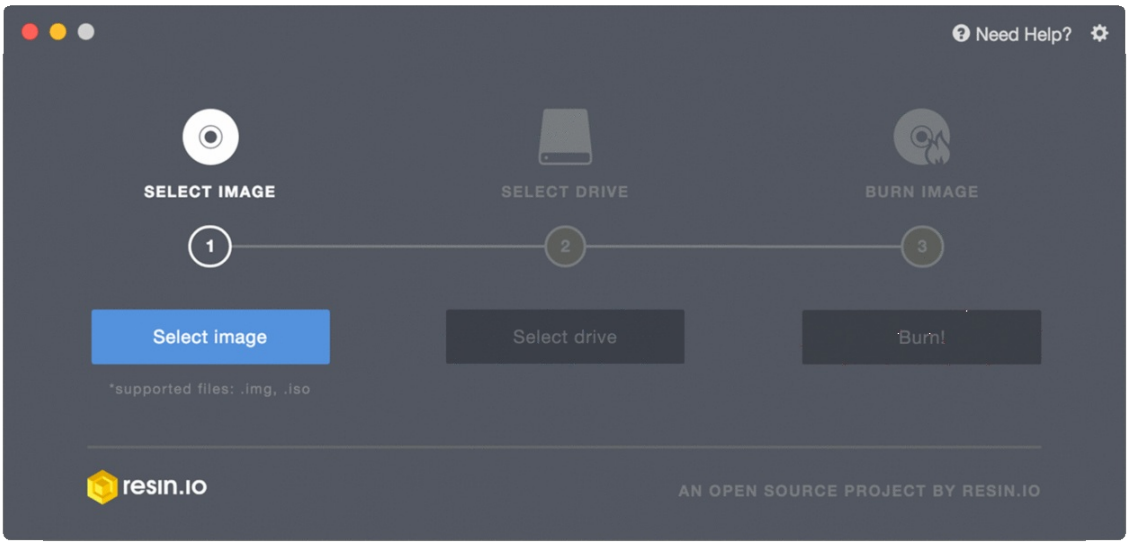
The .zip file you have downloaded need to be unzipped to get the image file for writing to your SD card.

Write the operating system image to the SD card

With the image file corresponding to your Poppy creature, you need to use an image writing tool to install it on your SD card. First unzip the image you have previously downloaded.

Burn the image with etcher (GUI software)

Download and install etcher. It works for Windows, OSX, and GNU/Linux operating systems.



Select the unzipped image, select the SD-card or MMC drive, press the *burn* button, and wait until it is done.

Now you are ready to assemble your robot!

Burn the image with `dd` (CLI software)

> This method works only for GNU/Linux and OSX operating systems, and is not recommended if you don't understand what you do.

- Run `df -h` to see what devices are currently mounted.
- If your computer has a slot for SD cards, insert the card. If not, insert the card into an SD card reader, then connect the reader to your computer.
- The new device that has appeared is your SD card. The left column gives the device name of your SD card; it will be listed as something like `/dev/mmcblk0p1` or `/dev/sdd1`. The last part (`p1` or `1` respectively) is the partition number but you want to write to the whole SD card, not just one partition. Therefore you need to remove that part from the name (getting, for example, `/dev/mmcblk0` or `/dev/sdd`) as the device for the whole SD card. Note that the SD card can show up more than once in the output of df; it will do this if you have previously written a Raspberry Pi image to this SD card, because the Raspberry Pi SD images have more than one partition.
- Run `umount /dev/sdd1` , replacing `sdd1` with the device name of your SD card (including the partition number).
- In the terminal, write the image to the card with the command below, making sure you replace the input file `if=` argument with the path to your `.img` file, and the `/dev/sdd` in the output file `of=` argument with the right device name.

> You will lose all data on your hard drive if you provide the device name of another running partition.

Make sure the device name is the name of the whole SD card as described above, not just a partition of it; for example `sdd` , not `sdds1` or `sddp1` ; or `mmcblk0` , not `mmcblk0p1` .

If you are running a GNU/Linux OS:

```
sudo dd bs=4M if=poppy-ergojr.img of=/dev/mmcblk0
```

If you are running OSX or another BSD based OS:

```
sudo dd bs=4m if=poppy-ergojr.img of=/dev/rdisk2
```

> The `dd` command does not give any information of its progress and so may appear to have frozen; it > could take more than five minutes to finish writing to the card. To see the progress of the copy operation you can run `sudo pkill -USR1 -n -x dd` in another terminal.

- Run `sync`; this will ensure the write cache is flushed and that it is safe to unmount your SD card.
- Remove the SD card from the card reader.

Now you are ready to assemble your robot!

Install Poppy softwares

> If you want to install the board of a tangible robot, go to the startup chapter instead.

This section will guide you to install Poppy softwares on your personal computer. It is useful **only** if you are in one of these situations:

- You want to control a simulated robot.
- You want to control a Poppy creature from your computer **without** using the embedded board (Odroid or Raspberry Pi).

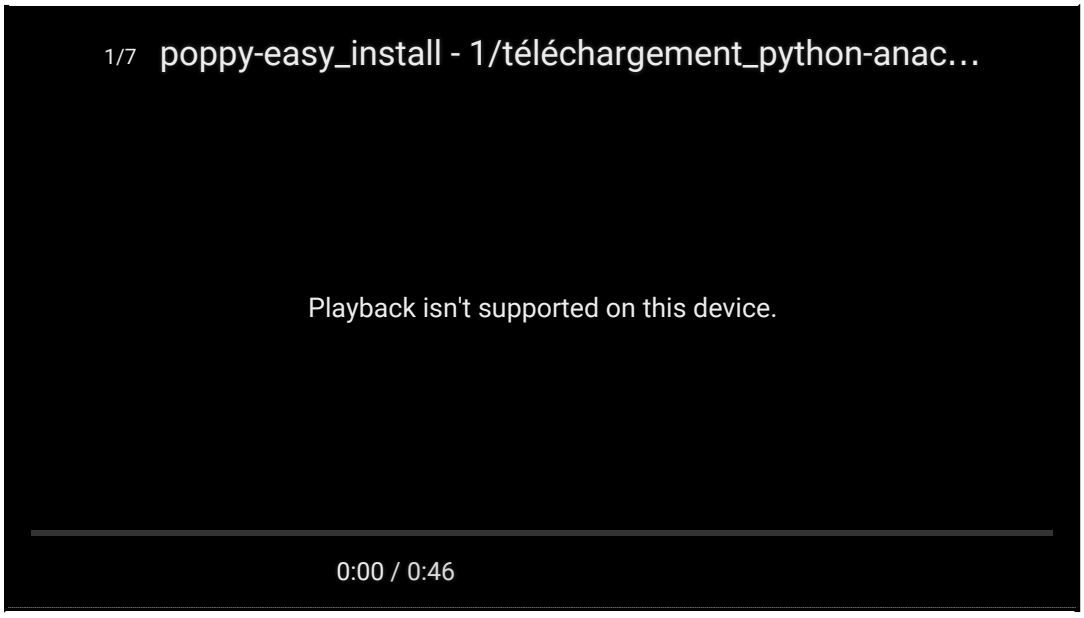
Poppy creatures are run by Python computer code. Depending on your operating system you will have to install Python and in any case you'll have to install the required software libraries. If you are getting started with Python and want to install a full Python environment for scientific computing, **we suggest you to use Anaconda Python distribution**.

> Poppy software libraries works in Python 2.7 and Python 3.3+. However as we develop them in Python 2.7, so it is less likely bugs for this Python.

Table of content:

Install Python and Poppy software on Windows

If you want a step by step screencast of the installation of Anaconda on Windows, you can see these videos (this is a YouTube playlist).



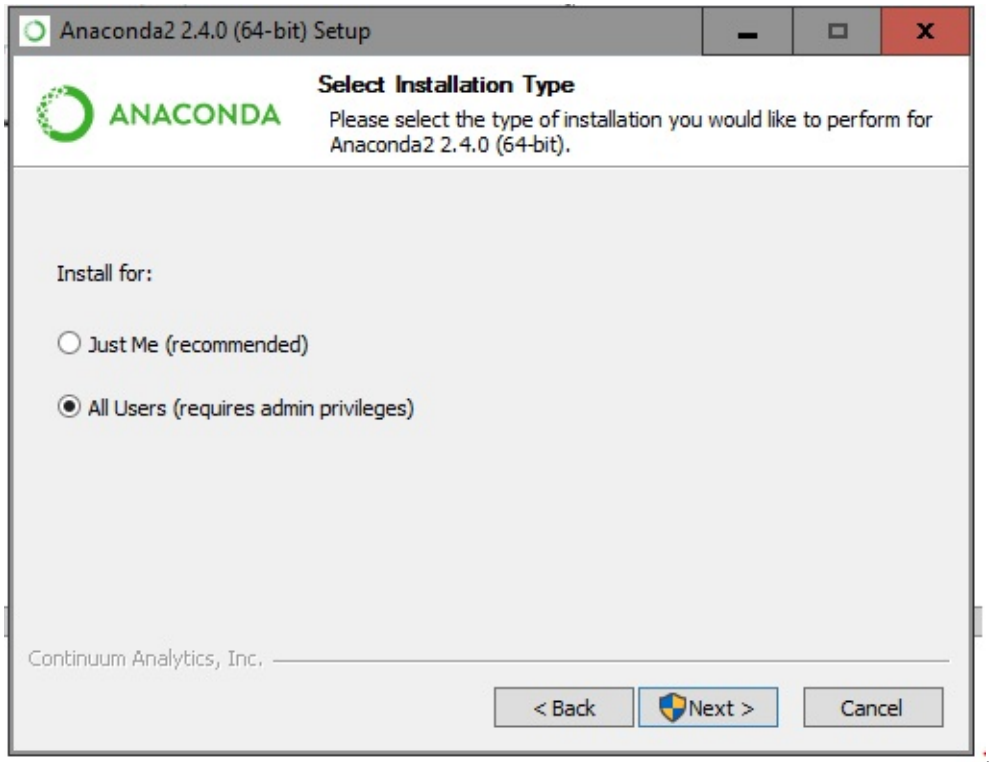
Install Python

We encourage the use of the Anaconda Python distribution. However, if you already installed a Python distribution like Canopy (shipped with scientific packages), you can directly install Poppy software.

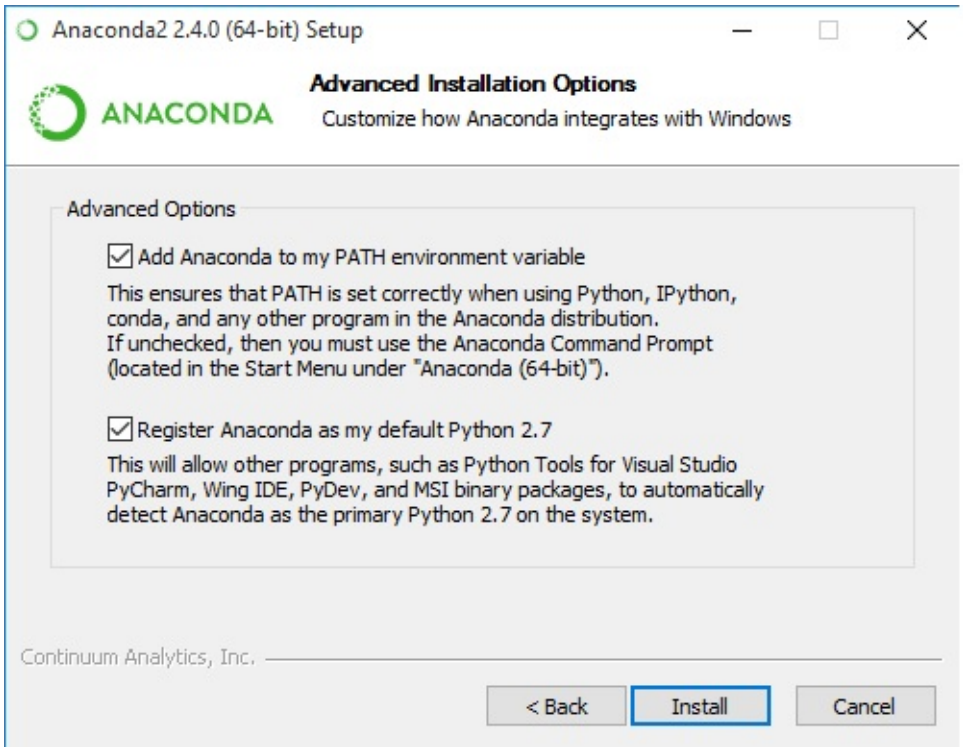
Anaconda

Download Anaconda Python distribution (400 MB) here for 64-bit computer or here for 32-bit.

Install it by clicking on "next" at each step. If you intend to install Anaconda for all users of your computer, be sure to select "all users".



It is also very important that the two check-boxes of the PATH and the default Python are checked.



Now you have a Python distribution ready to install Poppy software.

Miniconda (alternative to Anaconda)

Miniconda is a "light" version of Anaconda which contain only Python and the conda package manager. You can install it **instead of Anaconda** and save a lot of disk space (25 Mo vs 400 Mo), but you will have to do another step in the install process, and you will not have Jupyter notebook shortcut on the desktop. Download miniconda here for 64-bit computer or here for 32-bit computer.

Install it and be sure that the two check-boxes of the PATH and the default Python are checked.

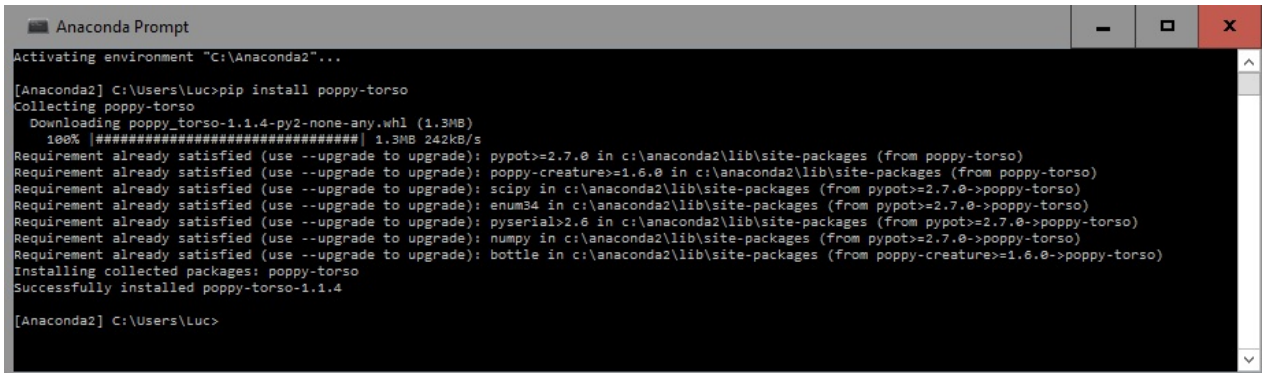
Open the Command Prompt (press the windows key and type "Command Prompt"), type and press Enter to execute the command below:

```
conda install numpy scipy notebook jupyter matplotlib
```

Now you have a Python distribution ready to install Poppy software.

Install Poppy software

Open the prompt of your Python Distribution (called *Anaconda Prompt* for Anaconda) or the *Command Prompt* of Windows, type and press Enter to execute the command below:



.

```
pip install poppy-torso --user -U --no-deps
```

This will install everything necessary to control a Poppy Humanoid. Substitute "poppy-torso" with "poppy-humanoid" or "poppy-ergo-jr" to install respectively a Poppy Humanoid or a Poppy Ergo Jr.

In case of update, it is advised to upgrade pypot (the motor library control) and the creature package separately:

```
pip install pypot --user -U --no-deps
pip install poppy-torso --user -U --no-deps
```

Install Python and Poppy software on Mac OSX

Mac OSX has a Python distribution installed by default. Before installing Poppy software, you need to install the Python package manager **pip**. Open a terminal copy and press enter to execute the command below:

```
curl --silent --show-error --retry 5 https://bootstrap.pypa.io/get-pip.py | sudo python
```

You can now install Poppy software for the creature of your choice:

```
pip install poppy-torso --user -U --no-deps
```

Substitute "poppy-torso" with "poppy-humanoid" or "poppy-ergo-jr" to install respectively a Poppy Humanoid or a Poppy Ergo Jr.

Install Python and Poppy software on GNU/Linux

Most of GNU/Linux distributions, have already a Python distribution installed by default.

Using the default Python distribution

Pypot, the main library of the robot is depending (amongst some other) on two big scientific libraries *Numpy* and *Scipy* which are themselves depending on C and Fortran code. These libraries may be installed with the Python package system (pip), but because of the huge number and differences between GNU/Linux distributions pip is not able to distribute binaries for Linux so all dependencies must be compiled... The solution to avoid the compilation of numpy and scipy is to install them with your distribution package manager.

On Ubuntu & Debian:

```
curl --silent --show-error --retry 5 https://bootstrap.pypa.io/get-pip.py | sudo python
sudo apt-get install python-numpy python-scipy python-matplotlib python-dev
```

On Fedora:

```
curl --silent --show-error --retry 5 https://bootstrap.pypa.io/get-pip.py | sudo python
sudo yum install numpy scipy python-matplotlib
```

On Arch Linux:

```
curl --silent --show-error --retry 5 https://bootstrap.pypa.io/get-pip.py | sudo python
sudo pacman -S python2-scipy python2-numpy python2-matplotlib
```

You can now install Poppy software.

> The downside is the Python libraries from you distribution system are very often out of date.

Using Anaconda (or miniconda)

If you want to have up to date numpy, scipy and ipython without having to compile them, we suggest you to install Anaconda or at least the conda package manager distributed with miniconda. Download miniconda (64-bit) with these command below in your terminal:

```
curl -o miniconda.sh http://repo.continuum.io/miniconda/Miniconda-latest-Linux-x86_64.sh
```

If you have a 32-bit computer

```
 curl -o miniconda.sh http://repo.continuum.io/miniconda/Miniconda-latest-Linux-x86.sh
```

Execute commands below and follow the instructions to install miniconda:

```
chmod +x miniconda.sh
./miniconda.sh
```

You can now install some required and other useful dependencies for Poppy software with conda:

```
conda install numpy scipy notebook jupyter matplotlib
```

You can now install Poppy software.

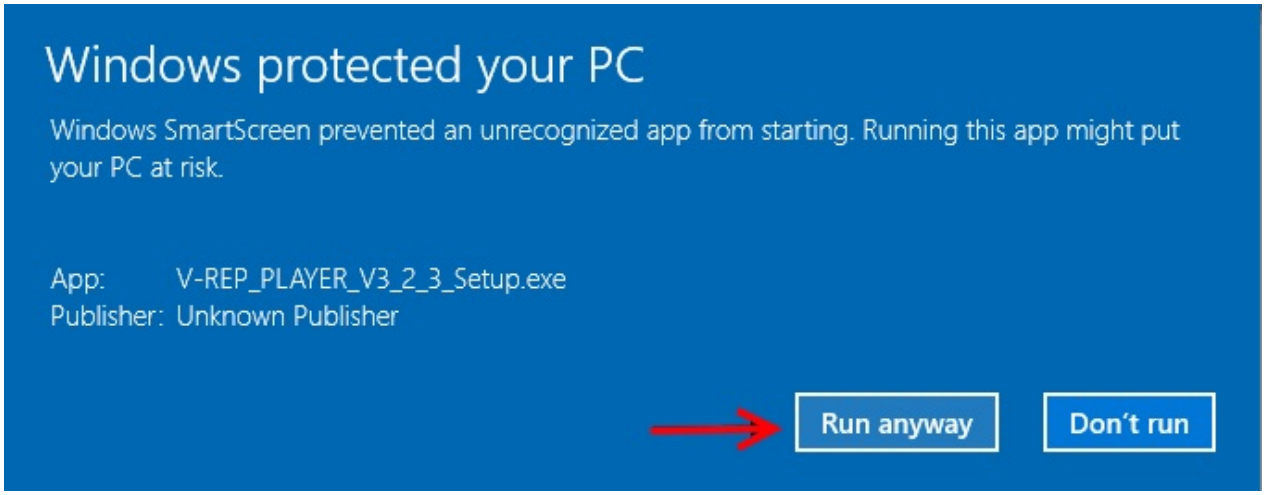
Install the robotic simulator V-REP

> You need to install Poppy softwares before installing the V-REP simulator.

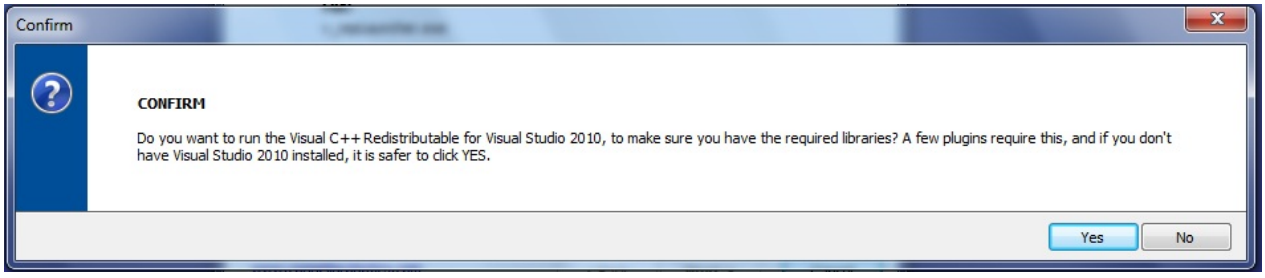
V-REP is an efficient robotic simulator mainly open source (GNU GPL), which is distributed under a free license for educational entities and have a commercial license for other purposes. There is also an *PRO EVAL* version which limit the right to backup. As you don't need to backup the scene to use V-REP with pypot (the Python library made for Poppy creatures), we suggest you to install this version to not worry about copyright infringement. If you want to modify the V-REP scene for adding or customizing a Poppy creature, you will have to use the PRO or the EDU version (look at the educational license).

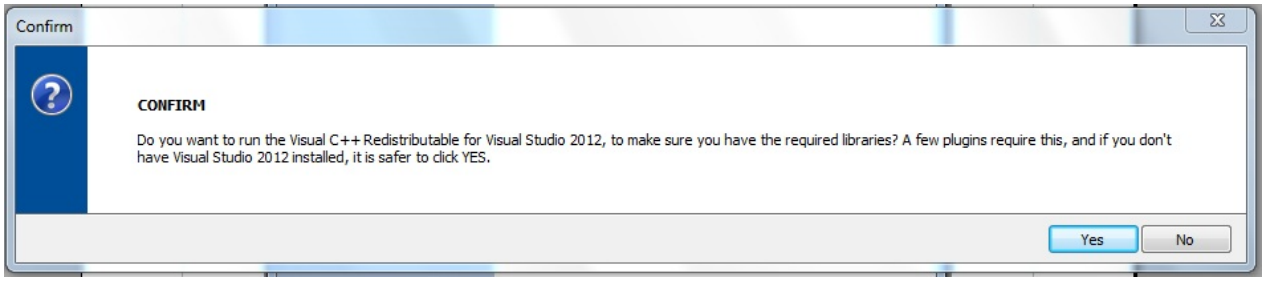
Install V-REP on Windows

Download V-REP PRO EVAL or EDU (if you are an educational entity). As V-REP is not signed, you will have to pass the Windows SmartScreen (on Windows 10) popup to begin the installation.

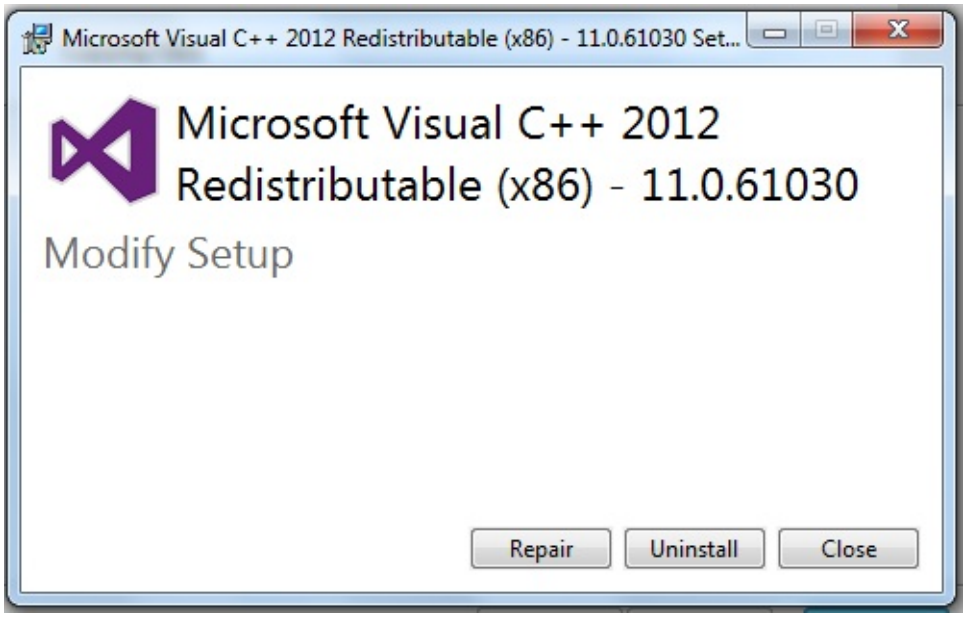


During the installation, make sure to install *Visual C++ Redistributable 2010* and *Visual C++ Redistributable 2012*.





Even if you already have *Visual C++ Redistributable 2010* or *Visual C++ Redistributable 2012*, it is advised to "repair" them (it is a re-installation process).



After the installation you can test if V-REP works well.

Install on MAC OSX

> This paragraph is not currently written. Your help is welcome to fulfill it !

Install on GNU/Linux

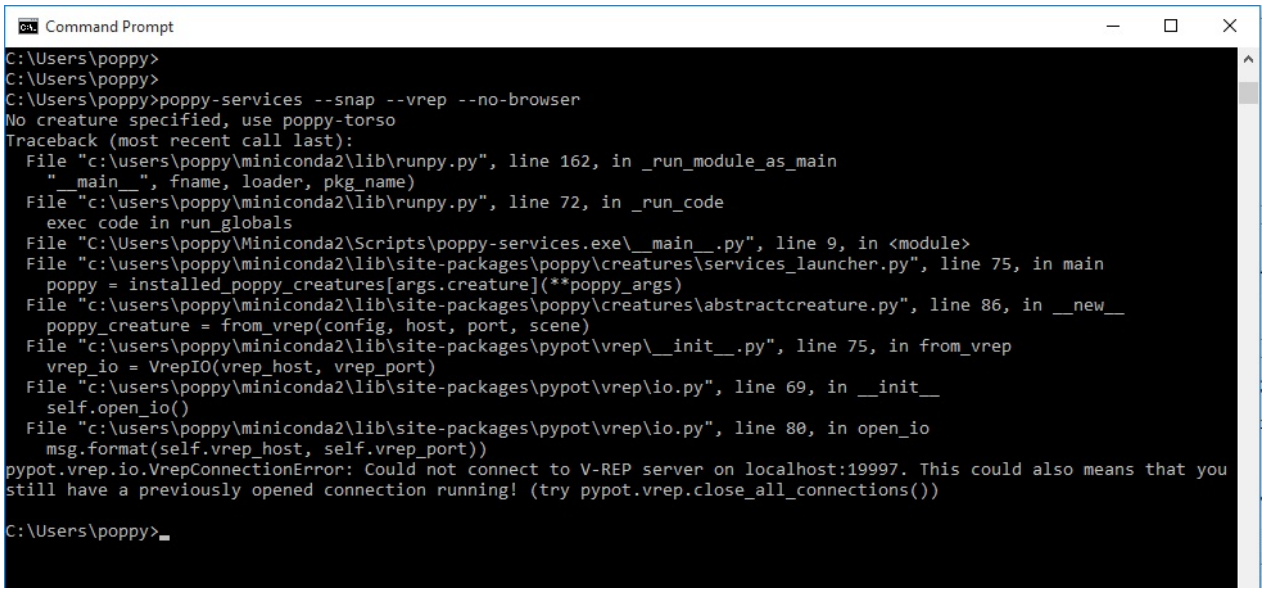
> This paragraph is not currently written. Your help is welcome to fulfill it !

Test your installation

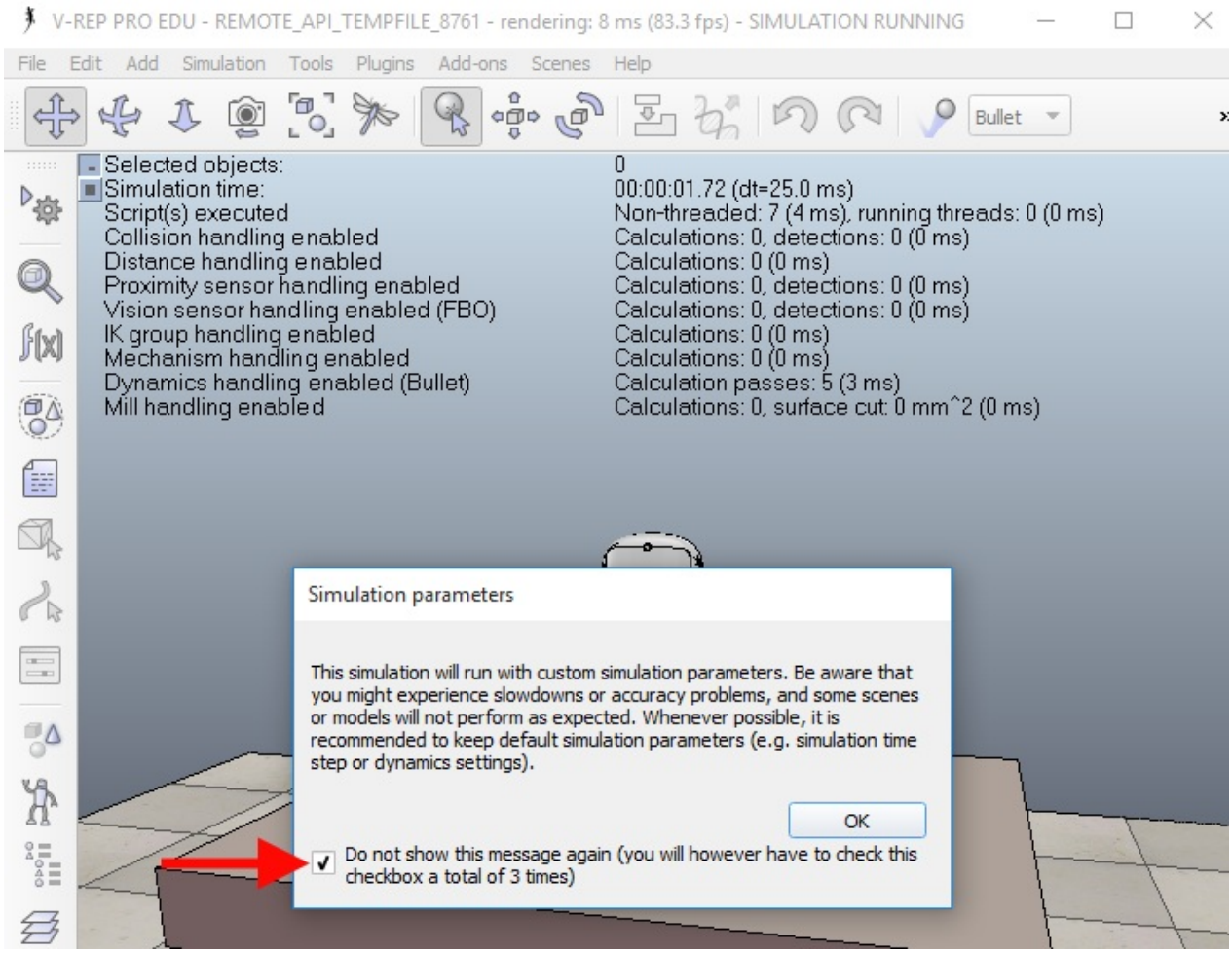
Open V-REP with a double click on the desktop icon. Open the prompt of your Python Distribution (called *Anaconda Prompt* for Anaconda) or the *Command Prompt* of Windows, type and press Enter to execute the command below:

```
poppy-services --snap --vrep --no-browser poppy-torso`
```

After a one or two seconds, you will have an error like the picture below in your Command prompt.



If you switch to the V-REP window, a popup appeared to inform you that the simulation use custom parameters. This popup block the communication to the Python API of V-REP. **You have to check the check-box "Do not show this message again" and press "Ok".**



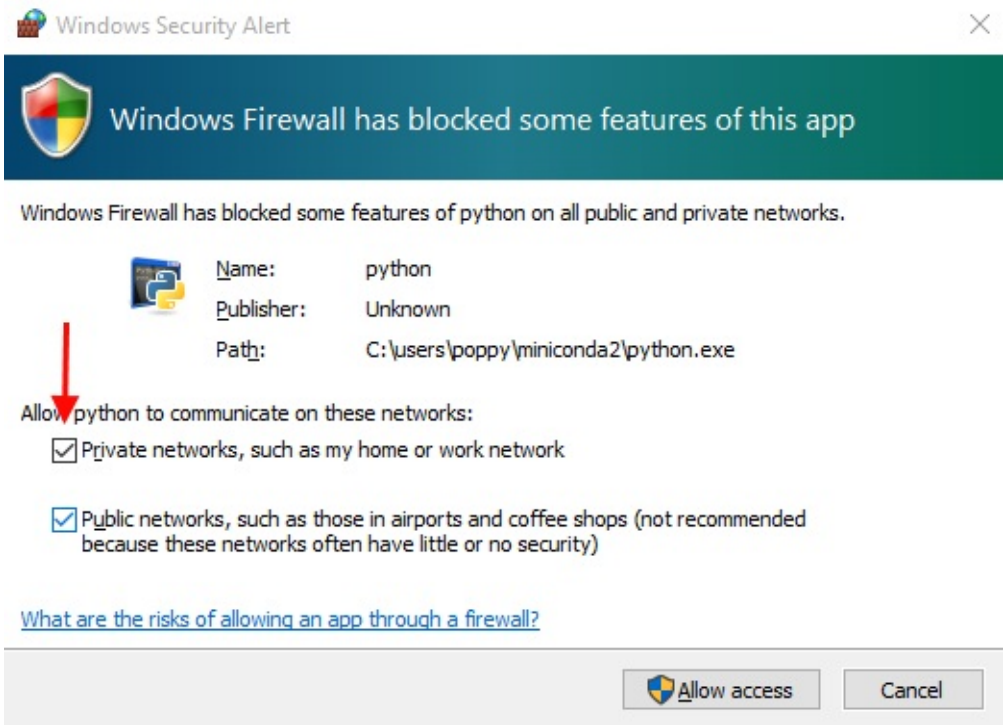
Switch the the command prompt window. You will have to execute the last command (`poppy-services --snap --vrep --no-browser poppy-torso`) and click again to the V-REP popup (with the check-box checked). **This process will have to be done *three times* to make it works well!**

> To avoid retyping the same command again and again, you can press the up arrow key to call the last typed line.

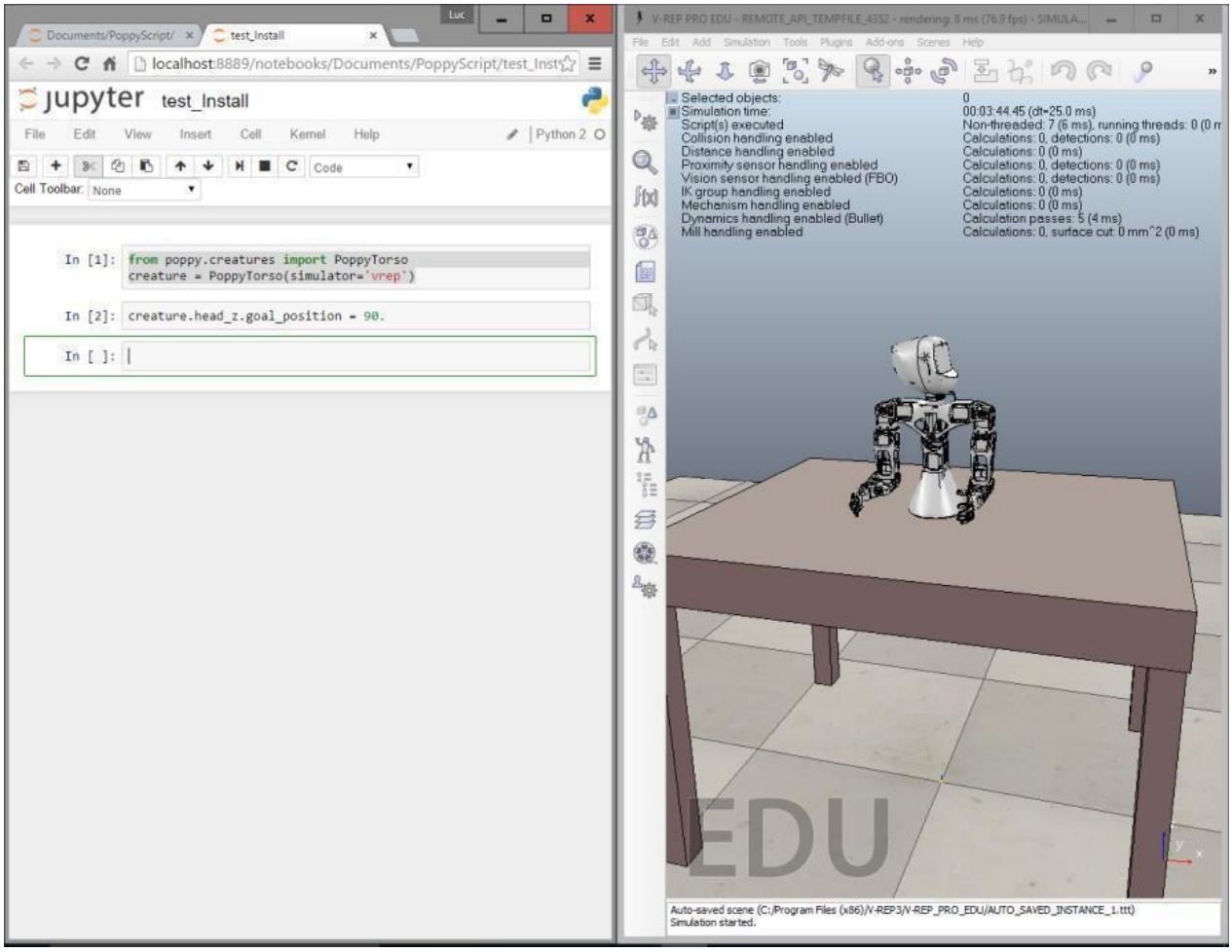
When the setup of V-REP is ready, you can execute the last command without the "--no-browser" part.

```
poppy-services --snap --vrep poppy-torso
```

If you see a firewall popup like the picture below, be sure to check the "private network" check-box.



If everything works, a new tab have been opened on your default web-browser. You can program you robot in Snap*!* or in Python.



Install drivers

> This chapter is only for people who want to control a tangible robot **without** an embedded board (Raspberry Pi or Odroid). **It is a special case for advanced users.**

If you intend to control tangible robots from your computer **without** a Raspberry Pi or a Odroid, and you use a computer with Windows (vs GNU/Linux or MAC OSX), you may need to install manually drivers for the USB2AX or the USB2Dynamixel.

If you use a USB2AX

If the USB2AX is not recognized out of the box (its LED stay red after having been plugged) on your computer, you probably need to install manually its drivers. The installation process and the files to download can be found on the USB2AX documentation. You don't need drivers for GNU/Linux or MAC OSX, but note that it doesn't works very well with MAC OSX.

If you want to control XL-320 motors (protocol Dynamixel v2) from an USB2AX you may need to update the firmware to the version 04 of the USB2AX.

If you use a USB2Dynamixel

You need to install FTDI drivers on your computer. You have to low the "Latency Timer Value" from 16ms to 1ms (minimum allowed value) as explained in the FTDI documentation to avoid pypot timeouts.

Install a Poppy board

> This chapter is only for people who want to create from scratch a Raspberry Pi or Odroid image. **It is strongly advised** to simply burn a pre-made system image on your robot.

To install a Poppy board, we start from a vanilla distribution (Debian or Ubuntu), remove some useless stuff and launch some scripts.

Keep in mind that our install scripts are not written for end users: it is not well maintained and there is almost no error messages. If you encounter issues with these scripts, you can post a message in the support section of the forums.

For a Poppy Ergo Jr / Raspberry Pi

Raspberry Pi are preformated with n00b, you need to install Raspbian first.

Download the image of your system:

- Raspbian Jessie if you are using a **Raspberry Pi 2**.

Write the image to the SD-card with you favorite disk writer tool as explained in the startup section.

> If you are using Windows, you have no native SSH client ; you have to download and install putty or mobaxterm to use SSH.

Login to the board in SSH: `ssh pi@raspberrypi.local` , password=raspberry.

You will need to make sure that you have enough free space in your raspberry. The easiest way is to use the raspi-config script to expand your partition to the full SD-card. Just log into your raspberry and run (you will need to reboot it afterwards):

```
sudo raspi-config --expand-rootfs
```

Be sure that your board is connected to the Internet, and use "raspoppy" installer:

```
curl -L https://raw.githubusercontent.com/poppy-project/raspoppy/master/raspoppyfication.sh | bash -s "poppy-ergo-jr"
```

Reboot after the end of the installation. The hostname, default user and password will be all set to "poppy" (`ssh poppy@poppy.local` password=poppy). You can test your installation with the web interface in your web browser http://poppy.local.

Install a Poppy Torso / Humanoid on a Odroid U3 or Odroid XU4

These boards come with a working Ubuntu base image on the MMC you can use the install scripts on it. In the case you have not a fresh installation you have download and burn default system images:

- Ubuntu 14.04 for Odroid U3
- Ubuntu 14.04 for Odroid XU3/XU4

To burn it on the MMC/SD-card, look at the startup section.

Now you have a clean and fresh installation, you can mount your memory card to your board, plug your ethernet connection, and power up.

Login to the board in SSH: `ssh odroid@odroid.local` , password=odroid.

> If you are using Windows, you have no native SSH client ; you have to download and install putty or mobaxterm to use SSH.

Be sure that your board is connected to the Internet, download and run poppy_setup.sh (replace 'poppy-humanoid' below with poppy-torso' if you want to install a Poppy Torso robot):

```
wget https://raw.githubusercontent.com/poppy-project/odroid-poppysetup/master/poppy_setup.sh -O poppy_setup.sh
sudo bash poppy_setup.sh poppy-humanoid
```

You should lose your ssh connection because of the board reboot. This reboot is needed to proceed to the finalisation of the partition resizing. Now your board should install all the poppy environment. **Please do not unpower the board or shut-it down.**

You can see the installation process by reconnecting you to your board with your new poppy account: `ssh poppy@poppy.local` password=poppy. **Because of the compilation of heavy Python packages (Scipy, Numpy) it can take more than 1 hour to complete.**

A process will automatically take you terminal and print the installation output. You can leave it with `ctrl+c`. You can get back this print by reading the install_log file:

```
tail -f install_log
```

Be patient...

At the end of the installation, your board will reboot again. You can look at the log `tail -f install_log`, if everything ended well, last lines should be:

```
System install complete!
Please share your experiences with the community : https://forum.poppy-project.org/
```

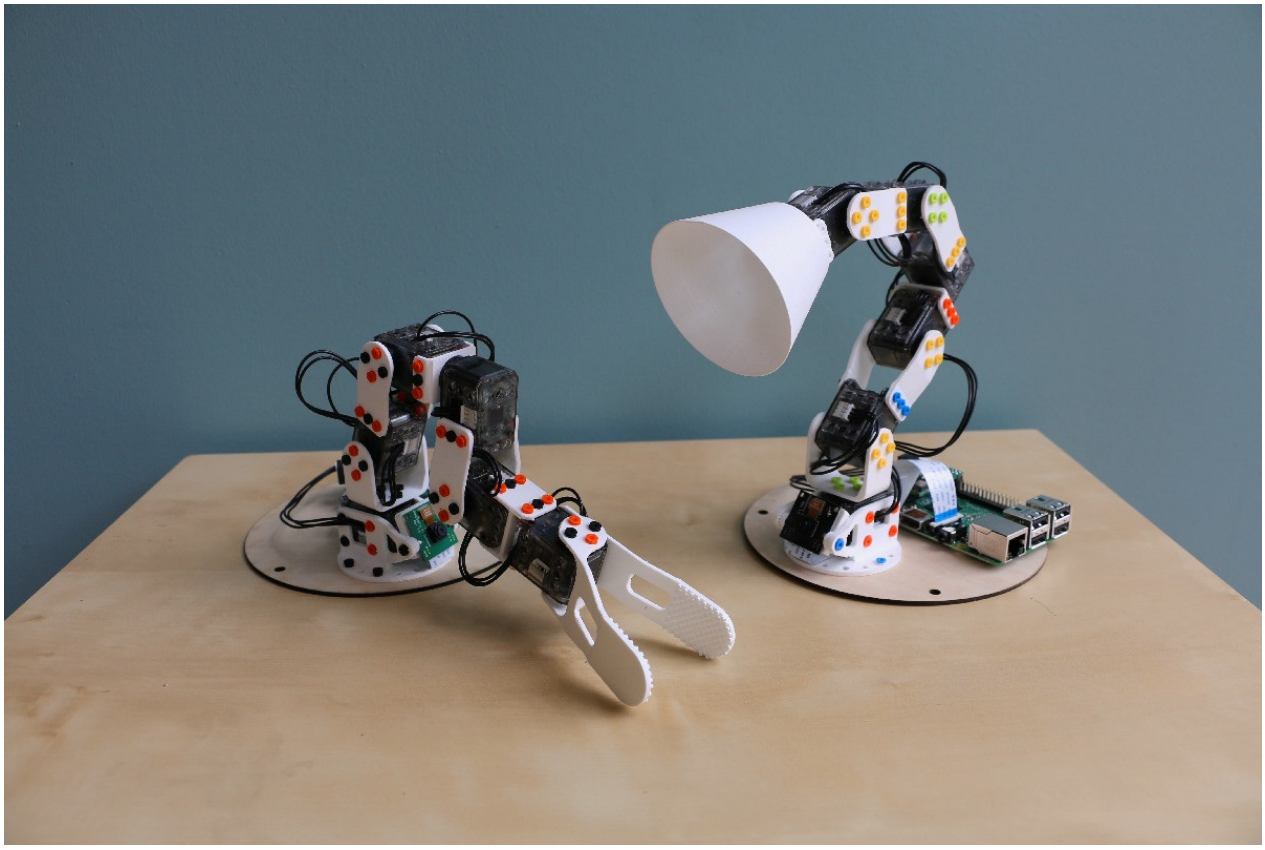
> If you are not sure of what going up, you can see if the install process is running with: `ps up $(pgrep -f 'poppy_launcher.sh')`

The hostname, default user and password will be all set to "poppy" (`ssh poppy@poppy.local` password=poppy). You can test your installation with the web interface in your web browser http://poppy.local.

Assembly guides

This section will provide step by step documentation of various libraries used in Poppy robots.

Assembly guide for the Ergo Jr



The Poppy Ergo Jr robot is a small and low cost 6-degree-of-freedom robot arm. It consists of very simple shapes which can be easily 3D printed. They are assembled via OLLO rivets which can be removed and added very quickly with the OLLO tool.

Its end effector can be easily changed. You can choose among several tools:

- a lampshade,
- a gripper,
- or a pen holder.

Thanks to the rivets, they can be very quickly and easily swapped. This allows the adaptation of the tooltip to the different applications you plan for your robot.

The engines have the same functionality as other Poppy creatures but are slightly less powerful and less precise. The advantage being that they are also less expensive.

The electronic card is visible next to the robot, which is very advantageous to understand, manipulate, and plug extra sensors. No soldering is needed, you just need to add the shield for XL-320 motors on top of the Raspberry Pi pins.

This chapter will guide you through all steps required to entirely assemble a Poppy Ergo Jr. It will cover:

- motors configuration
- electronic assembly
- hardware construction

The entire assembly should take about one or two hours for the first time you build one. With more practice, half an hour should be more than enough.

At the end of the process, you should have a working Poppy Ergo Jr, ready to move!

We recommend you to follow carefully the instructions. Even if the Ergo Jr can be easily disassembled, it is always disappointing to need to start from scratch again because you forget to configure the motors, a wire is missing, or a motor is reversed.

Bill of materials

Here you will find the complete list of material (BOM) needed to build a Poppy Ergo Jr.

Poppy's material

- 1x Pixl board *(electronic board to control XL320 motors)*
- 1x disk_support.stl (using laser cutting) the 2D plan can be found here. *You can also 3D print the base but it will take a lot of time*
- the 3D printed parts STL here
  - 1x base.stl
  - 3x horn2horn.stl
  - 3x side2side.stl
  - 1x long_U.stl
  - 1x short_U.stl
  - 1x support_camera.stl
  - the different tools
    - 1x lamp.stl
    - 1x gripper-fixation.stl
    - 1x gripper-fixed_part.stl
    - 1x gripper-rotative_part.stl
    - 1x pen-holder.stl
    - 1x pen-screw.stl

Robotis parts

- 6x Robotis dynamixel motors XL-320
- 1x set of OLLO rivets (about 70 colored and 4 grey)
- 1x OLLO TOOL

Screw

- 4x M2.5x6mm screw (for fixing the Raspberry Pi on the base)
- 4x M2x5mm screw (for fixing the camera)
- 4x M2 nuts (fixing camera)
- 1x Standoff Male/Female M2.5 10mm

Various electronics

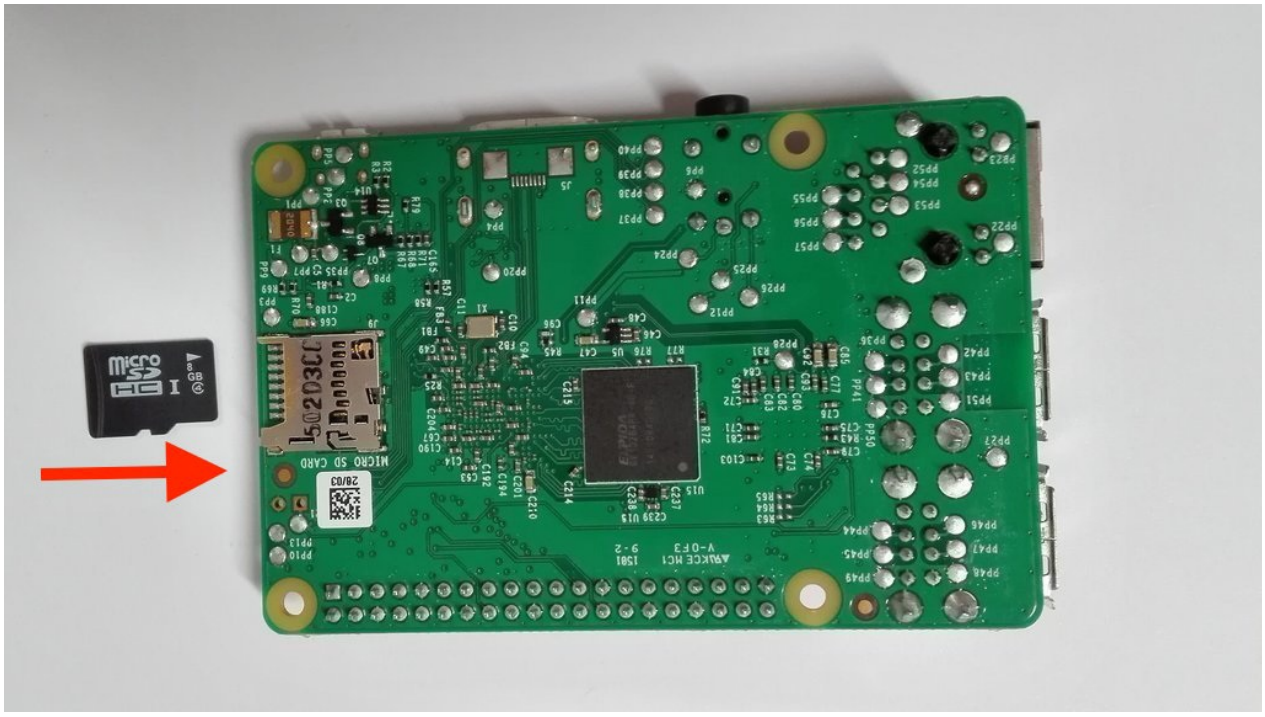
- 1x Raspberry Pi 2
- 1x micro SD 8Go
- 1x Rasperry Pi camera
- 1x AC power 7.5V 2A with a 2.1 x 5.5 x 9.5 jack connector (this one for instance).
- Short ethernet cable

Electronic assembly

Insert the micro-SD card in the Raspberry Pi

Make sure you use a pre-configured micro SD-card. If it not the case, you have to "burn" your micro-SD card with the ergo-jr ISO image, this is described in the startup section.

Insert the micro-SD card inside the Raspberry Pi: push the micro-SD in the connector slot until you hear a "click" sound.

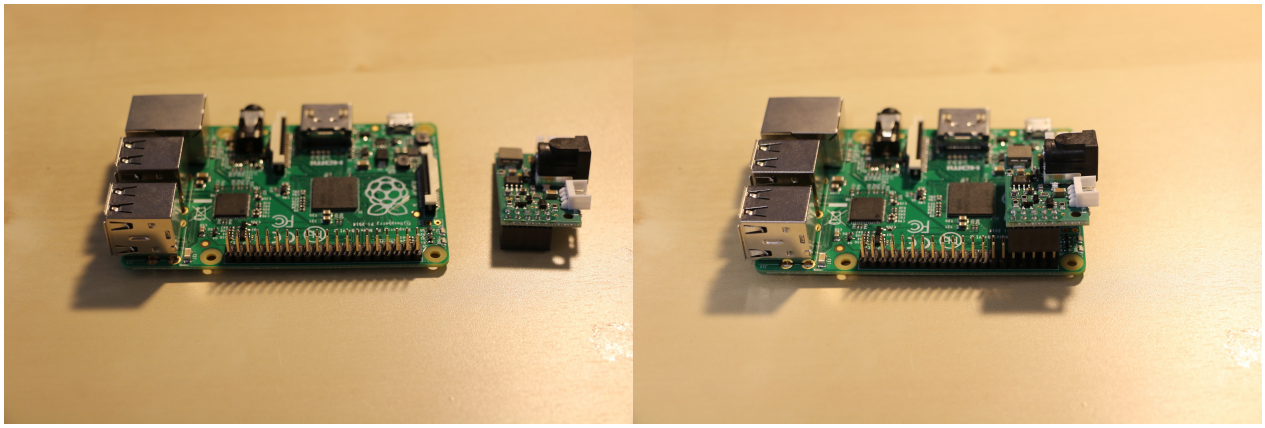


Assemble the pixl board

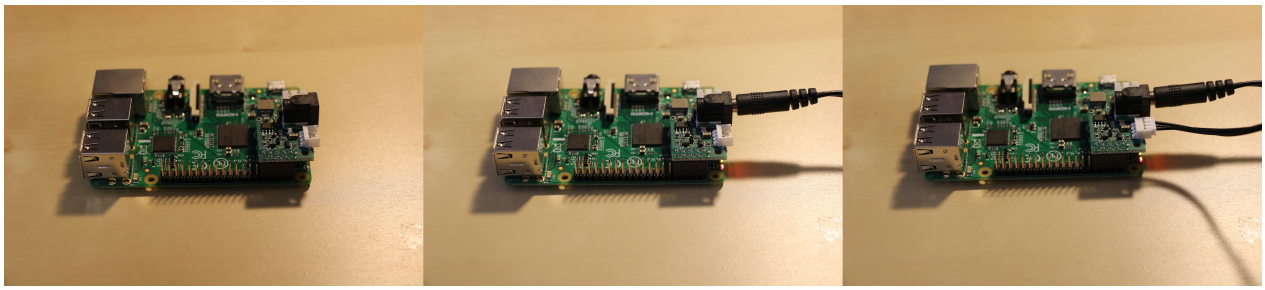
> The pixl board will be available for purchase in May 2016 at Generation robot.

Pixl bord allow to power up the Raspberry Pi from 7.5V DC power or battery and communicate with XL-320 motors.

Plug the pixl at the end of Raspberry Pi headers.



Once the pixl is plugged (**and not before**), you can plug the DC power and the motors wire.



> **You need to switch off the power supply** of the Pixl board before to put in or to take off the Pixl board of the Raspberry pi. Otherwise, you risk to burn the power converter of the Pixl board.

You can now configure your motors.

Motor configuration

The Ergo Jr is made of 6 XL-320 motors from Robotis. Each of this servomotor embeds an electronic board allowing it to receive different kind of orders (about position, speed, torque...) and to communicate with other servos. Therefore, you can chain up several of this servomotors and command them all from one end of the chain: each servomotor will pass the orders to the next one.

Yet, in order for the motors to be connected and identified on the same bus (same line), they must have a unique ID. Out of the factory they all set to the same ID: 1. In this section, we will give you details on how you can set a new and unique ID to each of your motors.

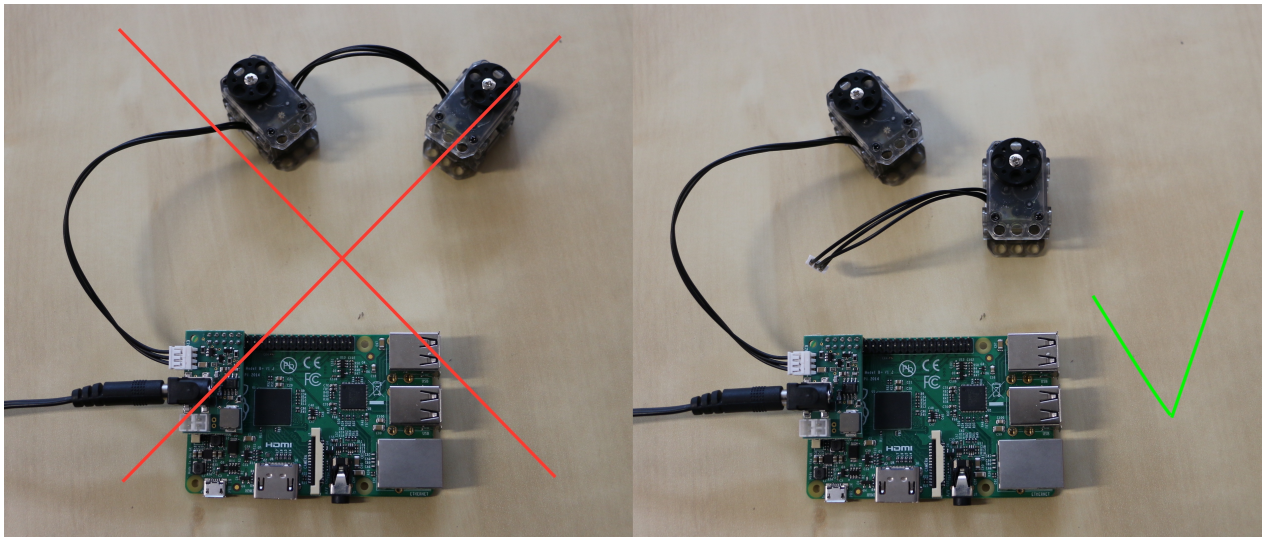
We recommend to configure motors in parallel of the hardware assembly. Meaning, that before assembling a new motor, you first configure it, then assemble to the rest of your robot. This will prevent you to swap motors. In the step-by-step assembly procedure, we will point out each time you need to configure a new motor. Furthermore, you will also be able to directly configure the motor from the assembly notebook interface.

Configuring motors one at a time

As explained above, all motors have the same ID by default. **Only one motor at a time should be connected to the data bus when you configure them.** Otherwise, it will not work as all motors connected will think that the order sent on the line is intended for them, they will all try to answer resulting in a big mess.

Your electronic setup when configuring a motor should look like this:

- the Raspberry Pi
- the shield on top and the AC plugged
- a wire from the shield to the motor you want to configure
- a ethernet cable going from the Raspberry Pi to your computer or your router



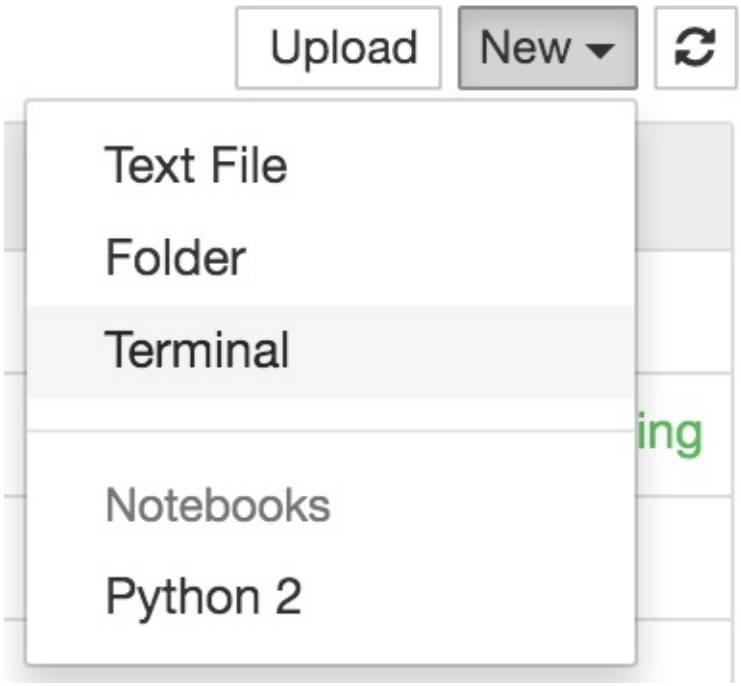
Web interface (user friendly)

> The web utility for configuring motors is still in the TODO stack.

Command-line utility

Robots come with a command line utility `poppy-configure` ; to use it you need to open a terminal on your Raspberry Pi.

You can access to the Raspberry Pi directly from your computer. To do so, open the page http://poppy.local in a web browser. You see the Poppy home page. Click on the "Python, Terminal" link and select "New Terminal".



Once the terminal is open, copy and press enter to execute the commande bellow.

```
poppy-configure ergo-jr m1
```

You have now configured the m1 motor of your robot. Once configured and that you see the message indicating that everything went well, you can unplug the motor (you don't need to turn off the card). The configuration of the motor is stored in the motor internal memory.

> Poppy Ergo Jr motors are named m1, m2, m3, m4, m5, m6. To configure the others motors, you have to change m1 by the name of the motor you want to configure in the command above.

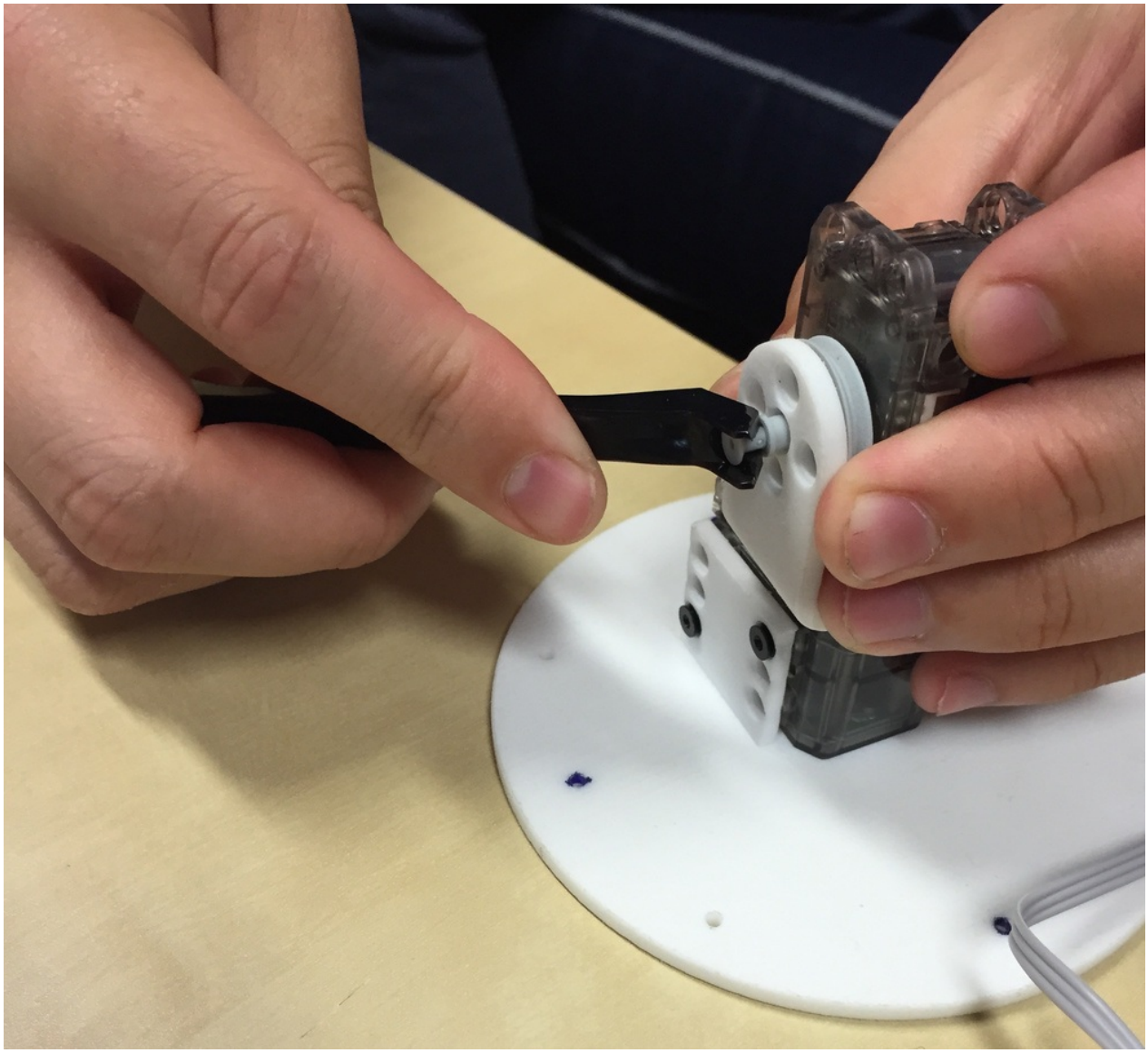
Mechanical Assembly

General advices and warnings

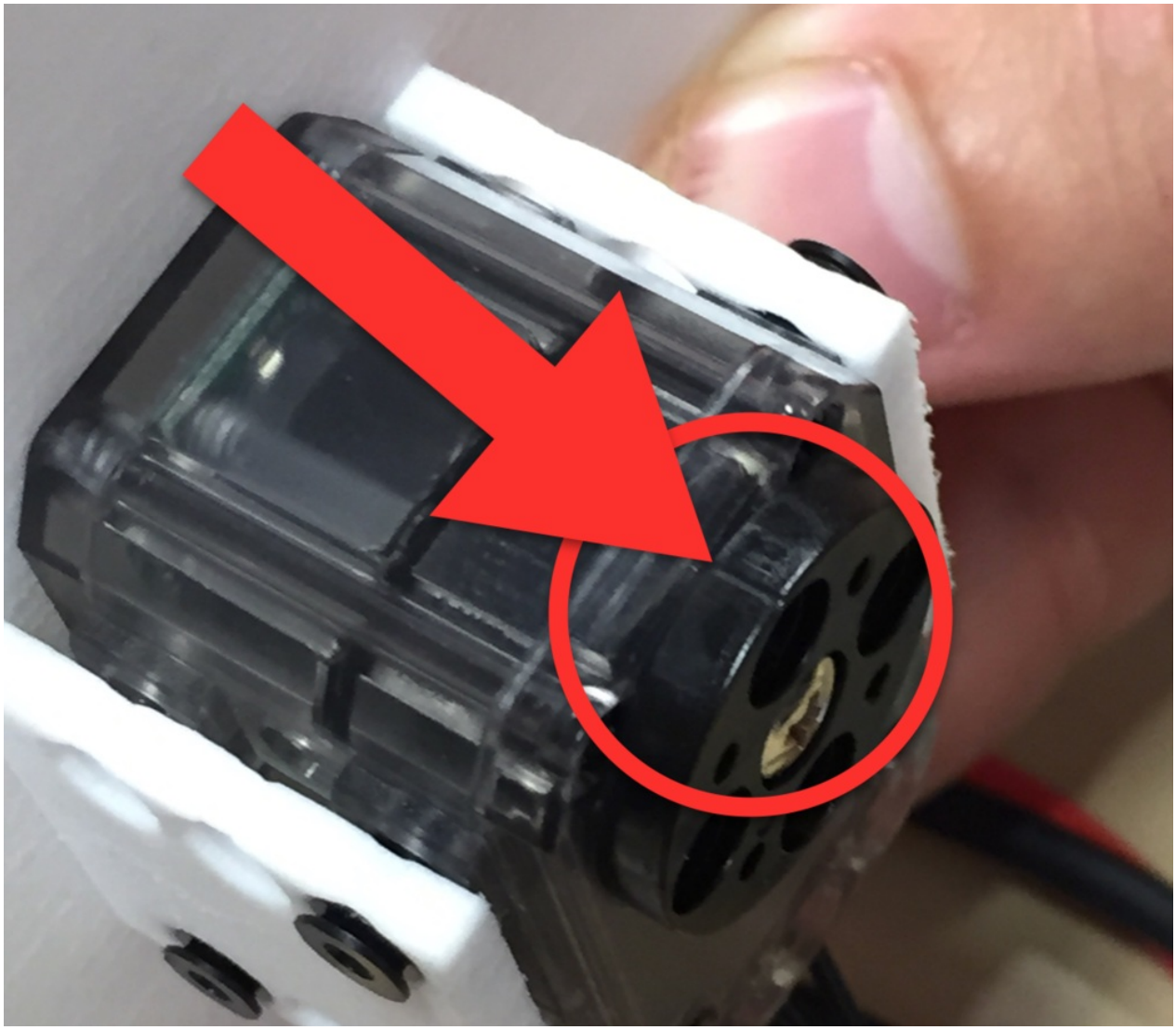
- You can assemble all the rivets before the construction. You have to put the edges of the first part in the second part holes. You will thus be able to remove them easily if needed.

- There are two kinds of rivets. The grey ones and the others. Grey rivets are longer to be able to be inserted in the motor axis, at the opposite side of the horn.

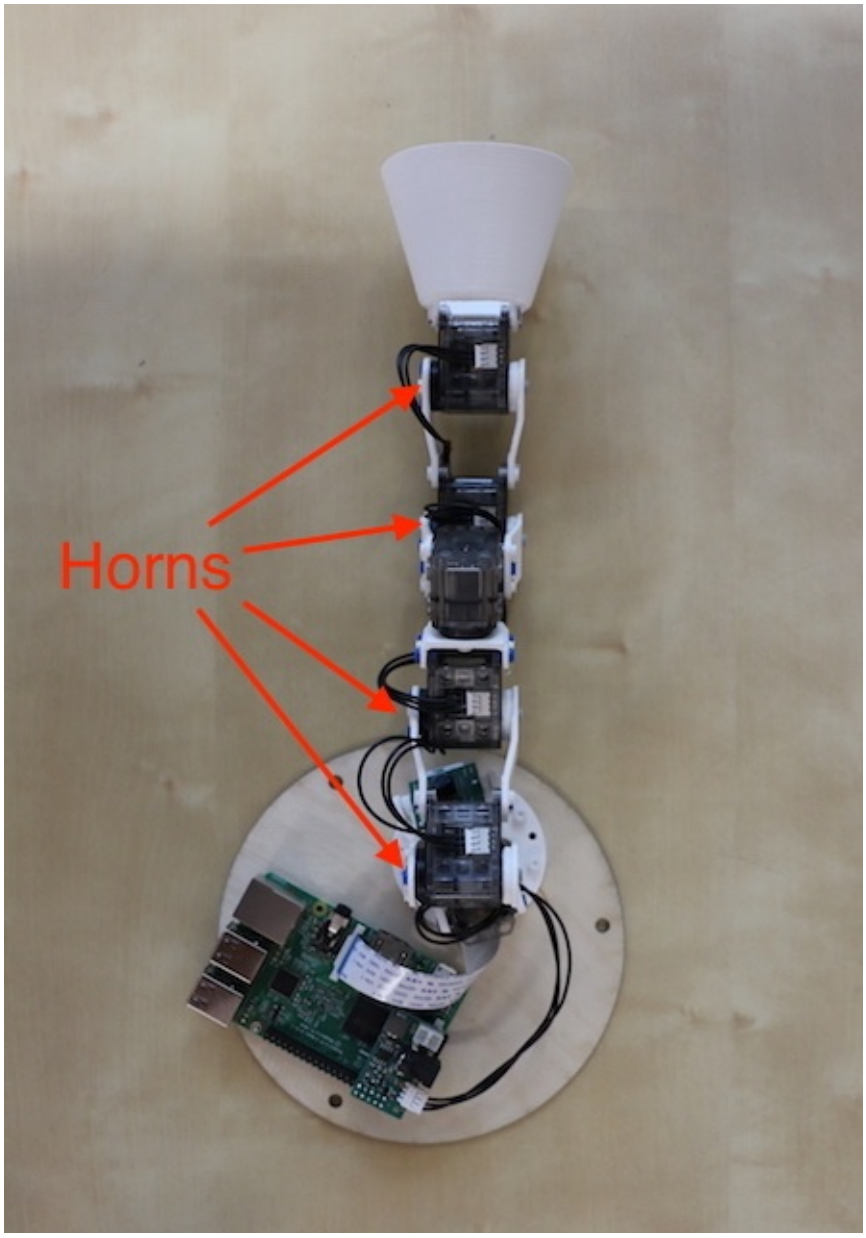
- Use the OLLO Tool for putting and removing rivets easily.



- Do not forget to put wires between motors while building the robot! Each motor, except the last, must have two wires; one connected to the previous motor and the other to the next (no favourite side).
- **Always align the horn with the motor before assembling them!** Otherwise your Poppy Ergo Jr will look all weird.



- Every motor horns (black revolving circle) are **facing the left side of the robot**. It is a convention but it will define the orientation of your motors.



Step-By-Step guide

Motor configuration (for all steps)

You can configure your motors before, during or after the the mechanical assembly but it is highly advised to configure each motor one by one in the construction order :

- configure motor m1
- assemble the base and motor m1
- configure motor m2
- ...

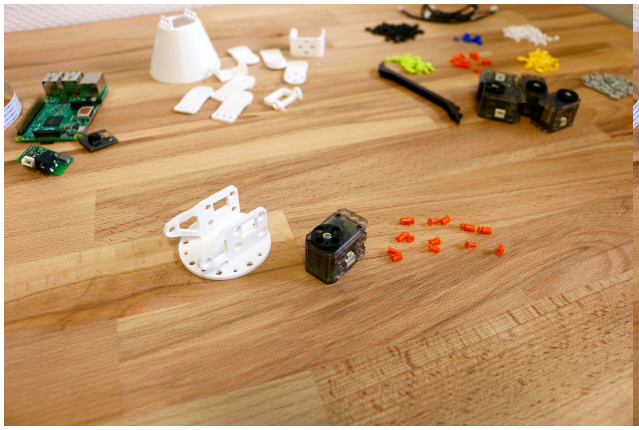
> To configure motors, you have to connect them separately one by one to the Raspberry Pi. If you try to configure a new motor wired to a previously configured motor, this will not work.

Please consult the motor configuration section for more informations.

Step 1

First, configure one XL-320 motor as "m1".

Mount the motor on the 3D printed base.




To do so, prepare 8 small rivets. put the first part in the second part without putting them in the motor. Then, place the motor in the base, with the horn facing up and near the more open side. Use the Ollo to grab a rivet between the first and the second part, then put the rivet in one the assembly holes. Once the rivet is in place, lock it by pushing the part 1 of the rivet in part 2.

Step 2

Configure the second motor, its name is "m2", with the following command in a poppy terminal:

```
poppy-configure ergo-jr m2
```

Mount the *long_U* part. Be careful wih the orientation of the U, the horn must be oriented in the left. Mount the motor "m2" on top of the construction.


Step 3

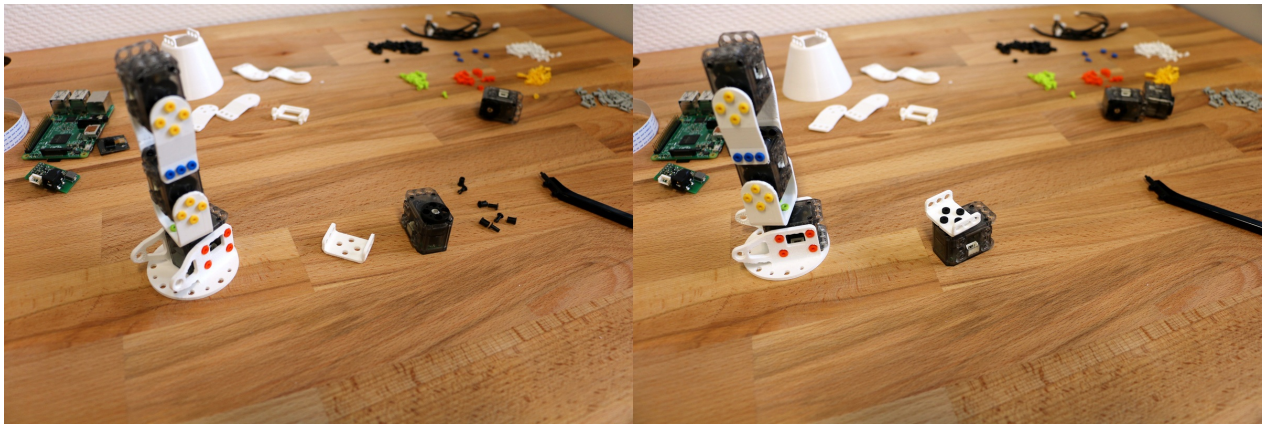
Configure a third motor: "m3".

Mount *horn2horn* and *horn2side* parts on motor "m2", and mount "m3" on top of the construction.

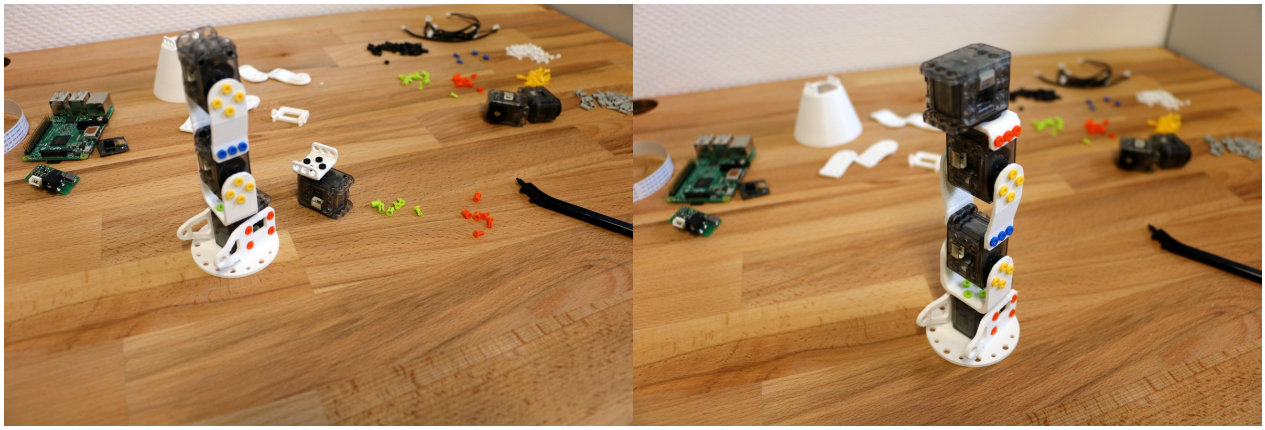
Step 4

Configure the fourth motor: "m4".

Mount the *short_U* on it.



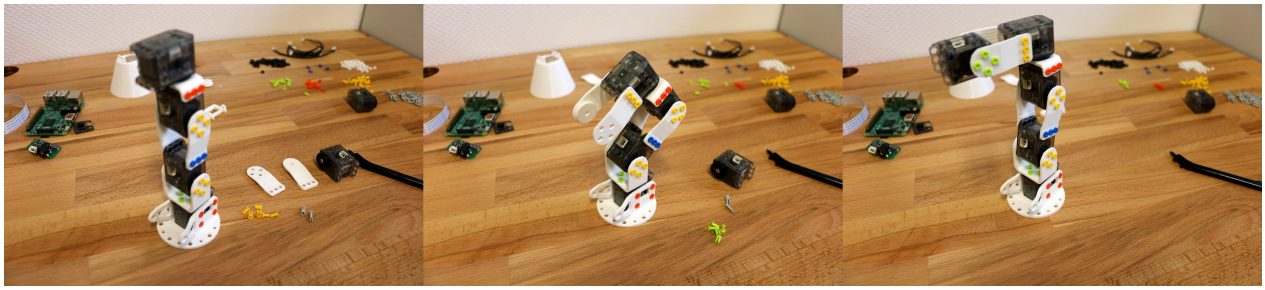
Mount motor "m4" and the assembled *short_U* on top of the previous assembly. The nose of the motor should be on the other side of the base.



Step 5

Configure the fifth motor: "m5".

Mount *horn2horn* and *horn2side* parts on motor "m4", and mount "m5" on top of the construction.



Step 6 - the tool of your choice

Configure the sixth motor: "m6".

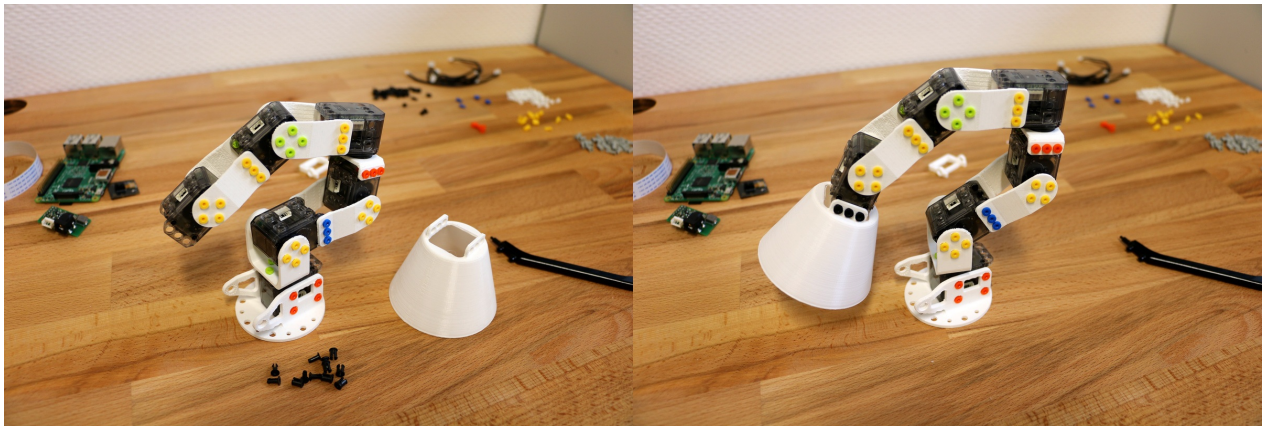
To finish your Ergo Jr, you need to add a tool at its end. So first choose the tool you want depending on what you want to do.

> Tools they can be easily and quickly changed, so you can adapt it to the different activities.

Lampshade or pen holder

Mount *horn2horn* and *horn2side* parts on motor "m5", and mount "m6" on top of the construction.

You can mount the pen holder or the lampshade on the motor "m6".



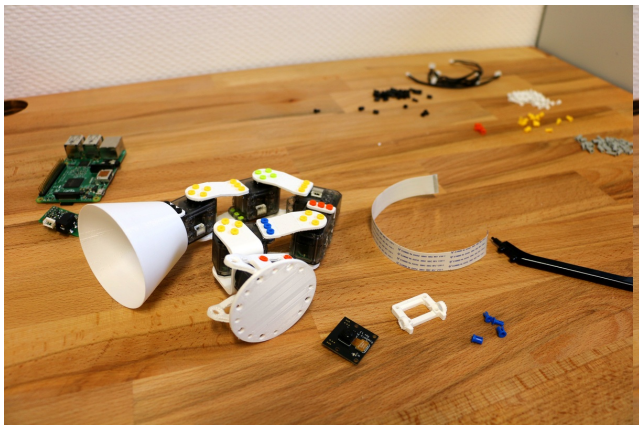
Gripper

Mount the *gripper-fixation* between motors "m5" and "m6".

Mount *gripper-fixed_part* and *gripper-rotative_part* on motor "m6".

Step 7 - electronics

Mount the support_camera part on the base. Fix the Raspberry Pi camera on it and move the camera flex cable between motor "m1" and the base.



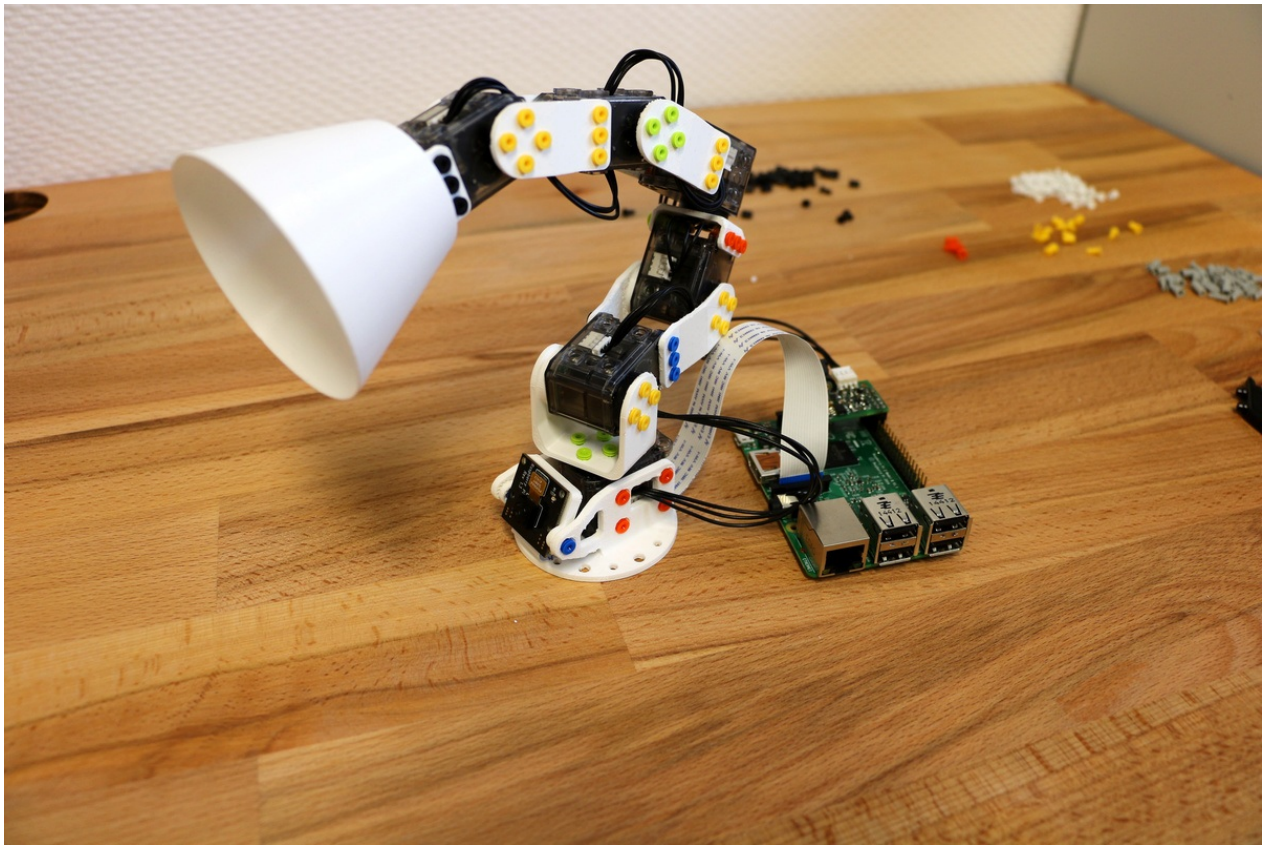
To fix the flex cable of the camera on the Raspberry Pi:

- open the camera connector by pulling on the tab
- make sure that connectors on the flex cable are facing away of the ethernet port
- push the flex on the port, and push the plastic tab down

Motors wires:

If it is not already done, you can plug every motors wires. Every motor has two connectors but there is no input or output: you just have to create a chain of motors. The first motor is linked to the pixl and the second motor; the last motor is linked only to the previous one, and every other motors are linked to the one above and ahead.

> Connectors of the motor "m1" (in the base) are a bit hard to link, you can use the OLLO tool to help yourself.

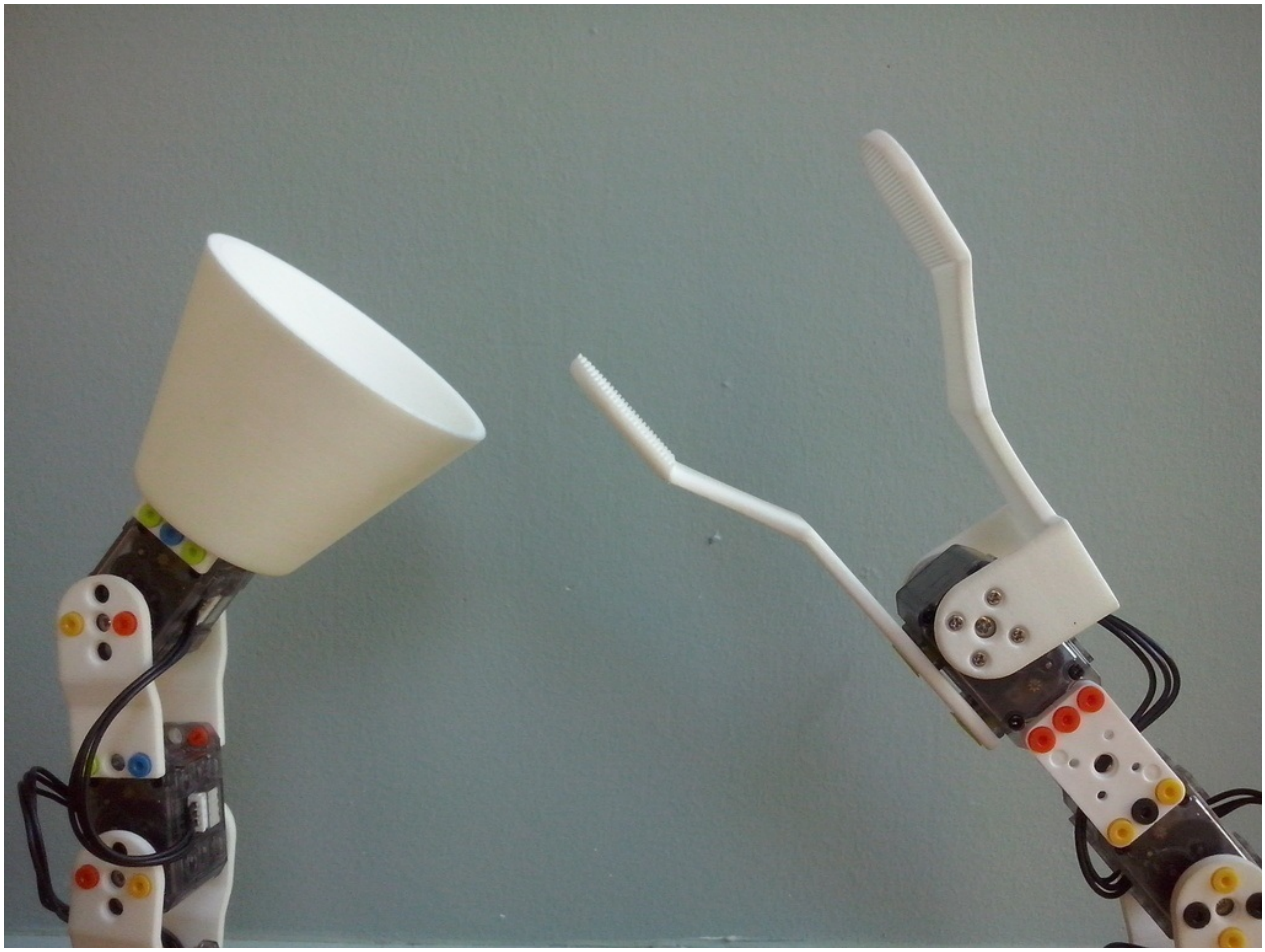


Step 8 - fix you ergo-jr to wood disk support

Mount you ergo-jr to the wood *disk-support*.

Mount the Raspberry Pi to the disk support, and use 4 x M2.5x6mm screw to fix it.

Done



Grab your favorite drink and relax.

Assembly guide for the Poppy Humanoid

> The assembly guide for the humanoid has still not been merged on the new doc. You can find it on the poppy-humanoid git repository.

Assembly guide for the Poppy Torso

> The assembly guide for the Poppy Torso has still not been merged on the new doc. You can find it on the poppy-torso git repository.

Programming

There is different way to program you Poppy robot. This choices depend of your programming skills and your hown preferences!

- If this is your first experience in programming you should probably start by Programming with Snap!.
- If you want to use python without any stark command line and editor your should look at the Usage of Jupyther notebooks section.
- If you want to lear how to use your robot through python look at the Programming in Python section.
- If you want to remote control your robot with external runing code look at the Robots APIs section.

Programming Poppy robots using Snap*!*

Snap*!* is a blocks-based graphical programming language that allows users to create interactive animations, games, and more, while learning about mathematical and computational ideas.

Snap*!* was inspired by Scratch (a project of the Lifelong Kindergarten Group at the MIT Media Lab), but also targets both novice and more advanced users by including and expanding Scratch's features.

Snap*!* is open-source and it is entirely written in javascript, you can use it from the official website but you can also use a copy of the website in your personal computer and open the `snap.html` file in your browser.

> Even if Snap*!* use JavaScript and HTML5 which are browser independent technologies, opening blocks for Poppy robots in Snap*!* is far faster in a web browser based on Webkit engine. We strongly recommend you to use Chromium Browser (which is very similar to Chrome without tracking tools), or Google Chrome.**

Introduction to Snap*!* programming

This chapter will focus on things necessary to understand in Snap*!* for using Poppy creatures.

If you want a well designed online lesson on Snap*!* we strongly encourage you to look at the "Beauty and Joy of Computing" (BJC) course made by the University of Berkeley for New York high school students.

Some of the snapshots and concepts of BJC have been used for writing this chapter.

Connect your robot to Snap*!*

If you use a simulated robot on V-REP

> You need to have installed Poppy software libraries and V-REP simulator on your computer. If it is not done, go to the install poppy software section.

First open V-REP.

The quickest way is to use the command line utility poppy-service. Copy and press enter to execute the command below in your command prompt (windows) or terminal (OSX and Linux):

```
poppy-services poppy-ergo-jr --snap --vrep
```

Substitute 'poppy-ergo-jr' with 'poppy-humanoid' or 'poppy-torso' to launch respectively a Poppy Humanoid or a Poppy Torso.

It will open a Snap*!* tab in your web browser for a simulated poppy-ergo-jr. If it is not automatically done, you can open Snap with preloaded blocks at simu.poppy-project.org/snap/

> Every popup in V-REP will block the communication to the robot interface. If a popup appear, close it and restart the command above.

Alternative method: Instead of using `poppy-service` you can start it in full python:

```
# use PoppyTorso PoppyHumanoid or PoppyEgoJr depending on what you want
from poppy.creatures import PoppyErgoJr
poppy = PoppyErgoJr(simulator='vrep', use_snap=True)
```

If you use a simulated robot on poppy-simu (web viewer)

> You need to have installed Poppy software libraries on your computer. If it is not done, go to the install poppy software section.

The quickest way is to use the command line utility poppy-service. Copy and press enter to execute the command below in your command prompt (windows) or terminal (OSX and Linux):

```
poppy-services poppy-ergo-jr --snap --poppy-simu
```

> poppy-simu is only available for poppy-ergo-jr. Other creatures are only supported in V-REP.

It will open a Snap*!* tab in your web browser for a simulated poppy-ergo-jr. If it is not automatically done, you can open Snap with preloaded blocks at **simu.poppy-project.org/snap/** and the robot viewer at **simu.poppy-project.org/poppy-ergo-jr**.

Alternative method: Instead of using `poppy-service` you can start it in full python:

```
from poppy.creatures import PoppyErgoJr
poppy = PoppyErgoJr(simulator='poppy-simu', use_snap=True)
```

If you have a tangible robot

First, you must be connected to the same network LAN area than your robot (e.g. on the same router or Wifi).

You have to go on the web homepage of your robot with its URL. You can use its IP address (for example http://192.168.1.42) if you have a way to know it or its hostname like http://poppy.local. To find its IP address look at the zeroconf chapter. To use directly its hostname http://poppy.local you must have a Zeroconf software installed on your computer (aka "Bonjour print services for Windows" if you are running Windows).

The home page of your poppy creature should look like the snapshot below:

Click on the "Start Snap!" link to open the Snap*!* interface at start the connection with the Poppy robot.

Poppy special blocks are stored in the Examples. Go to "file" icon -> open -> Examples -> click on "Poppy blocks". It may take some time to load the blocks (~5-15 seconds), be patient.

Interface and general ideas

Saving in Snap*!*

There are three ways of saving a project in Snap*!*

Save the project in your web browser

When you are not logged in Snap*!* Cloud, the default behaviour of Snap*!* is to save your project in **your browser**.

Technically this uses the Local Storage which is a memory space in your web browser where websites can store offline data. This is very convenient because you have not to register or to see Snap*!* project files, but keep in mind that **these projects are only visible in this specific web browser in this specific computer**.

Snap*!* Cloud

> « There is no Cloud, it's just someone else's computer ».

Instead of saving your projects on your web browser, you can save them in Snap*!* servers in UC Berkeley, called "cloud". Moreover, this allows you to share your project with anyone, with a simple link.

Create an account on Snap*!* cloud

Click on the cloud button -> "signup...".

Fill the required fields in the modal window for signing up.

You will soon receive a validation email with a random password. You can now log in with your username and password.

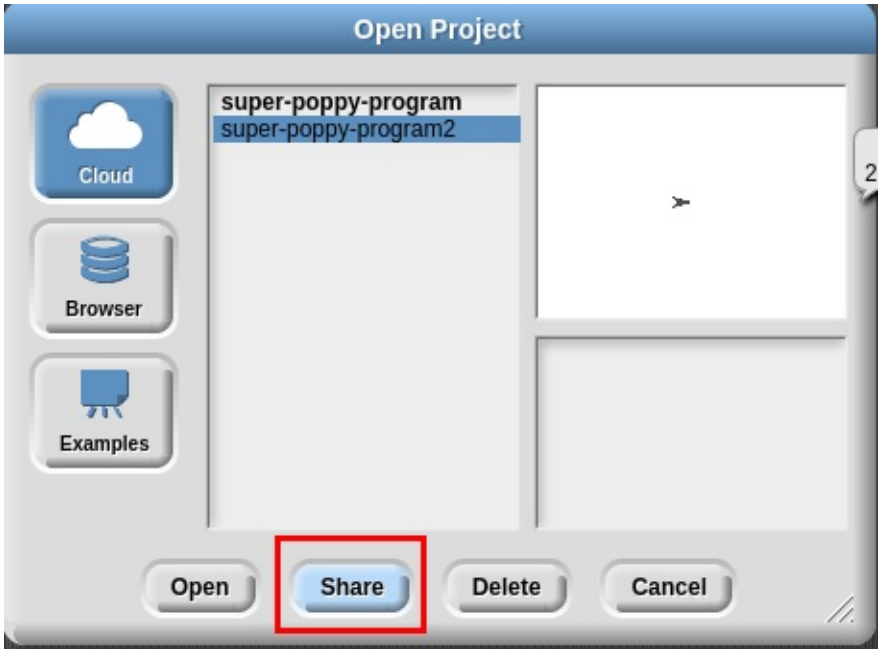
If you use your personal computer, remember to check the "stay signed in on this computer [...]" checkbox.

After logging in account, you are free to change your password: click on the cloud button -> "Change Password".

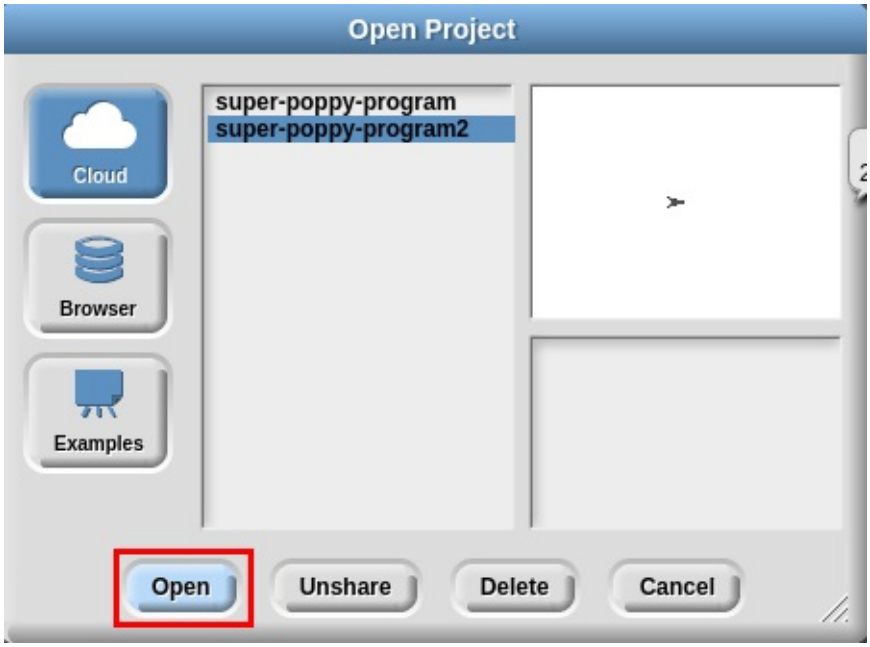
Share your Snap*!* project

The big advantage of using Snap*!* Cloud is the ability to share a copy of your project with anyone. To share a Snap*!* project, you first need to be logged in Snap*!* Cloud and having your current project saved ("save" or "save as"). Go to the "open" menu:

In the cloud section, select the project you want to share and click on "Share" button.



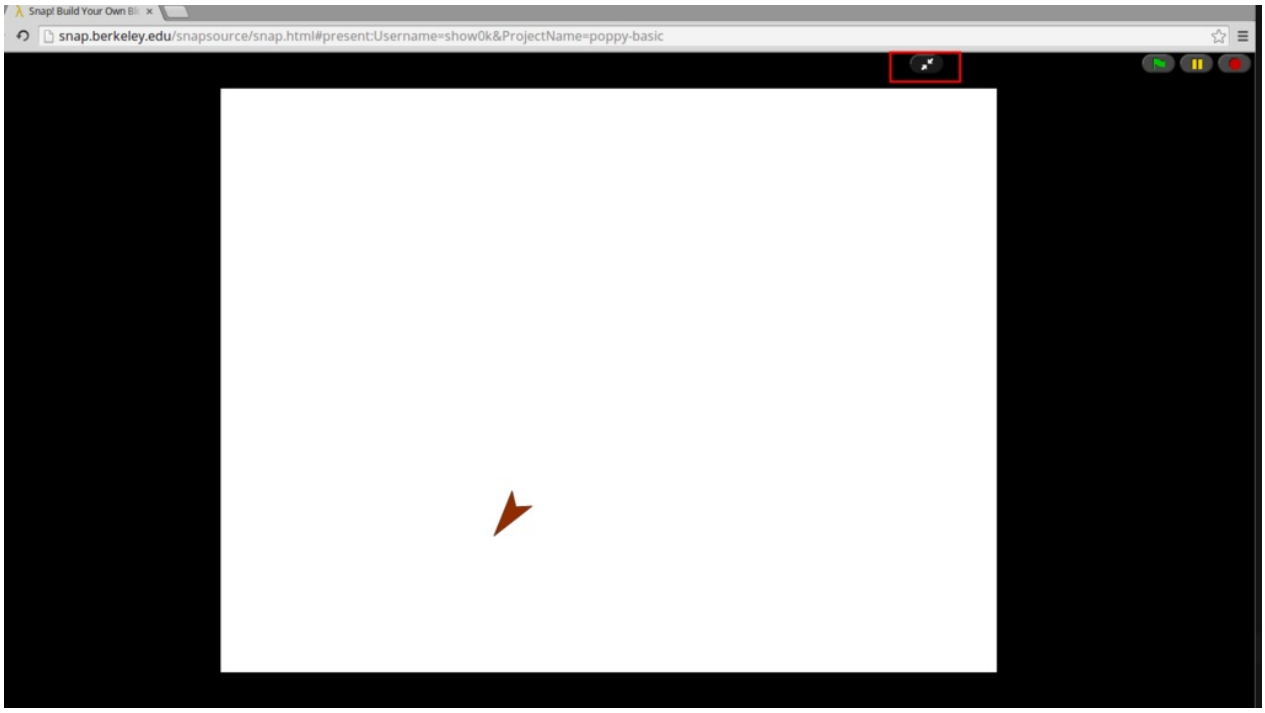
Here is the trick step: to see the share link, you have to click on the "Open" button.



And this will re-open your project with the public sharing URL.

You can copy and paste the URL and share it by the way you want to your friends or to the Poppy community with the forum forum.poppy-project.org.

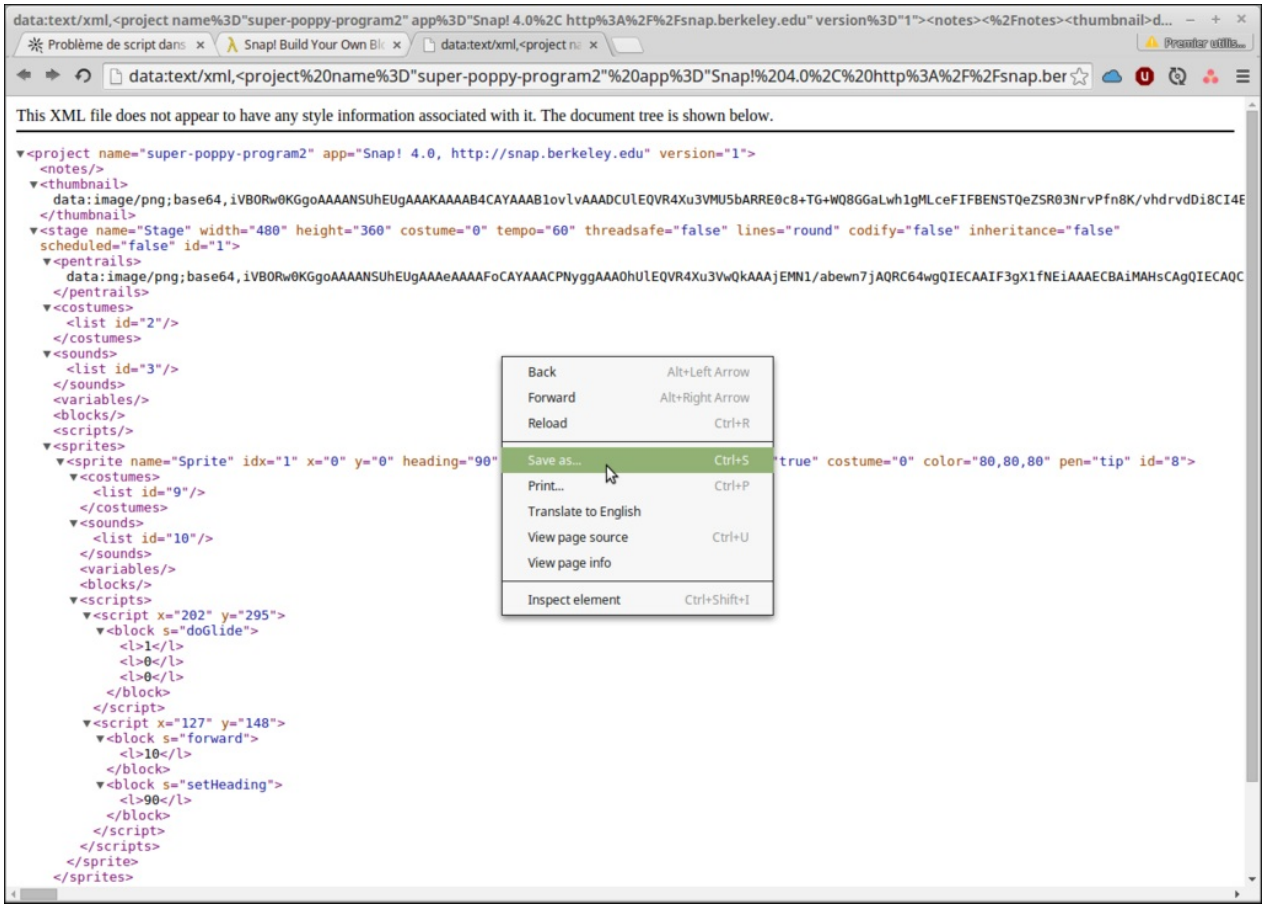
When you open a share project, the project is automatically opened in full screen on the sprite zone. To quit the full screen you have to click on the double arrow at the top of the snapshot below.



Export/Import your Snap*!* project

If you have a limited access to internet and you want to share project with other people, the best way is to export it:

A new tab in your web browser will be opened with an XML file like the picture below.



This file describe all your Snap*!* project in a a simple file. It's not made to be human readable so don't be afraid, you just have to save it on your computer. For that, do a right click, chose "save as" and a proper name and location on you computer for this project.

If you want to import a previously exported project, you simply have to click on the import section of the file icon.

Search Poppy blocks

Every Poppy block in Snap*!* begins by a robot icon. So you can search all of them by the robot keyword. To search a specific block, do a right click on the block area, or use the keyboard shortcut CTRL+F.

Network

First test your connection with the (tangible or simulated) robot with the "test connection" block.

. if the block answer is "You may have connection troubles", your "host" variable inside the Snap*!* project is probably wrong. The host variable must be the IP or the hostname+".local" of your robot ; if you're using V-REP localhost is used to point to your own computer.

Build your own blocks!

The functionality to build your own block was the first difference between Scratch and Snap*!* (now it's also possible to make custom blocks in Scratch)!

Description of Poppy blocks

Quick examples

Record and play back a move

You can record a movement on one motor and play it back

You can change the speed

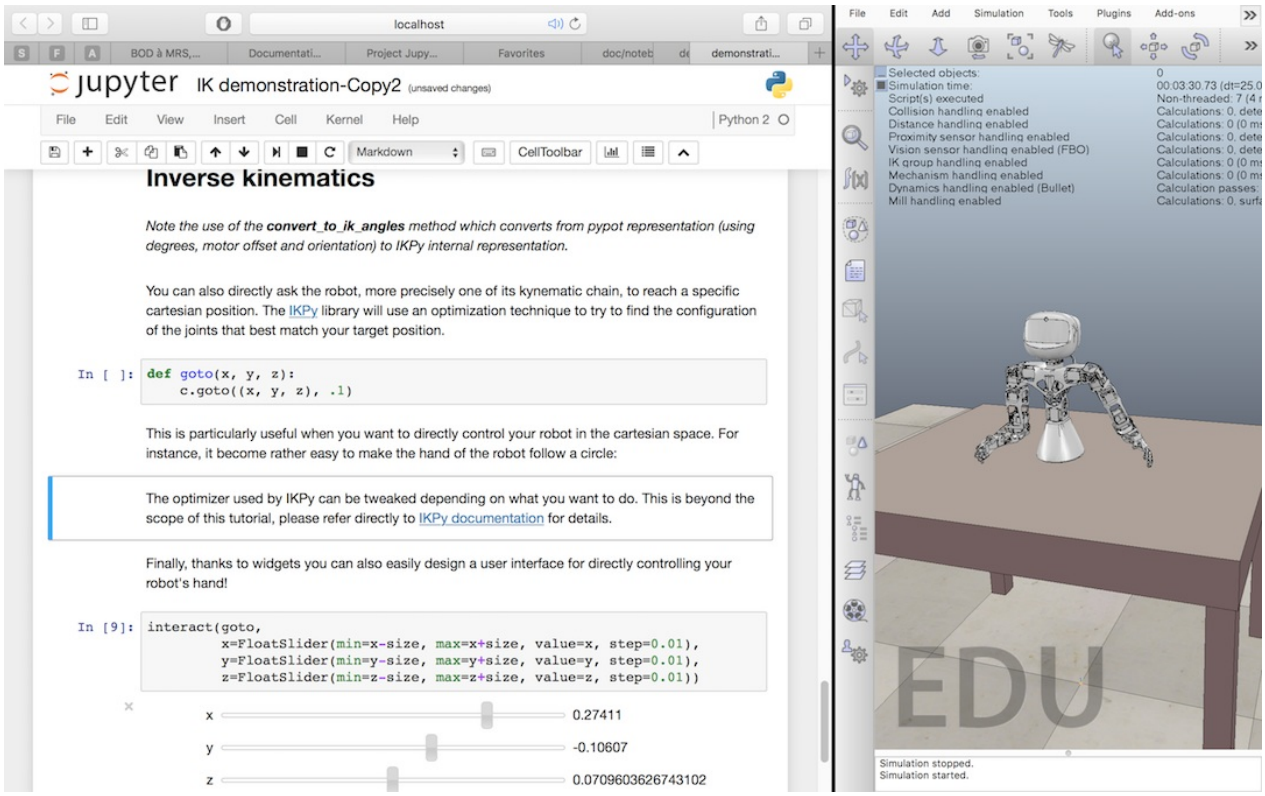
You can record it on all motors

or just a subset of your creature

Record and by demonstration movement

Jupyter Notebooks Gallery: using Python

Most of the existing examples of using Poppy robots in Python are given as Jupyter Notebooks. We strongly encourage the use of this web application as it allows "the creation and sharing of documents that contain live code, visualization and explanatory text". Furthermore, they also permit the design of interface for live controlling a robot thanks to widgets.



This chapter presents a gallery of notebooks and tries to organize them into different categories:

- Getting started
- Simulator
- HTTP API and remote connection
- Scientific experiments
- Education
- Going further (advanced topics)

For each notebook, we provide a short description of what it does, with which robot/simulator it can be used and of course a link. Most of the notebooks are written in english but you will also find some in french (and hopefully soon in other languages).

Getting started

- **Discover your Poppy Ergo Jr:** Notebook - Begin controlling your robot, launch behavior, send motor command, get values from the sensor...
- **Controlling a Poppy Humanoid in V-REP:** **Notebook** - Describe how to setup a Poppy Humanoid in V-REP and how to control it (motor control and sensor reading) from pypot in Python.
- **Record, Save, and Play Moves:** **Notebook** - Simple introduction on how to record by demonstration moves on any Poppy Creature. It also shows how they can be re-played and saved/load to/from the disk.

Notebooks en français

- **10 choses à savoir avec Poppy Humanoid/ErgoJr et V-REP:** **pour l'ErgoJr**, **pour l'Humanoid** - 10 informations de base pour bien commencer avec Poppy Humanoid ou Poppy ErgoJr simulés dans V-REP et comment les contrôler en Python.

Simulator

V-REP

- **Controlling a Poppy Humanoid in V-REP:** **Notebook** - Describe how to setup a Poppy Humanoid in V-REP and how to control it (motor control and sensor reading) from pypot in Python.
- **Interacting with objects in V-REP:** **Notebook** - Show how you can programtically add objects to the V-REP simulation and interact with them. This example uses a Poppy Torso but can be easily adapted to other creatures.
- **Learning the robot IK:** **Notebook** - Demonstrate how explauto can be used to learn the inverse kinematics of a Poppy Humanoid. The experiments are run in V-REP simulation but it also gives hints on how it can be transposed in the real world.

Notebooks en français

- **10 choses à savoir avec Poppy Humanoid/ErgoJr et V-REP** **pour l'ErgoJr**, **pour l'Humanoid** - 10 informations de base pour bien commencer avec Poppy Humanoid ou Poppy Ergo Jr simulés dans V-REP et comment les contrôler en Python.

HTTP REST API and remote connection

- **Controlling a robot using HTTP requests:** **Notebook** - Show how you can send HTTP requests to a robot, using the REST API, to control it. The notebook is based on a V-REP simulated Poppy Humanoid but can be adapted to other creatures.

Scientific experiments

Discover Explauto

- **Learning the robot IK:** **Notebook** - Demonstrate how explauto can be used to learn the inverse kinematics of a Poppy Humanoid. The experiments are run in V-REP simulation but it also gives hints on how it can be transposed in the real world.

Demo interface

- **Primitives launcher for Poppy Humanoid:** **Notebook** - Provides all codes needed to directly launched primitives (stand, sit, idle motions, limit torque...)

Education

Notebooks en français

Initiation à l'informatique en Lycée

- **Découverte:** **TP1**, **TP2**, **TP3** - Comprendre comment faire bouger simplement le robot. Utilisation des boucles. Ces TPs utilisent un Poppy Torso simulé dans V-REP.
- **Dialogue:** **TP1**, **TP2** - Établir un dialogue entre Python et le robot. Ces TPs utilisent un Poppy Torso simulé dans V-REP.

Going further

Low-level communicatio

Debug and setup

Benchmark

Extending Poppy software

Contributing to this gallery

Do not hesitate to let us know if some cool Notebooks are missing! You can directly send pull-request on GitHub or use the issue tracker.

Programming Poppy robots in Python

This chapter will guide you through how to control Poppy robots in Python. As it is actually the language used for writing Poppy core libraries, you will see how to access all the different levels of control, from the higher to the lower.

We will detail everything you need to know to directly program you robot using the Python embedded in the Poppy robot or to install everything locally. Note that this chapter does not intend to teach you Python or programming from scratch and thus if you are completely new to Python it may be good to start with a Python tutorial. Yet, we try to keep the tutorials as simple as possible and we will always warn you when some parts are targeting more advanced users.

We will try to provide as many examples as possible and point to the complete API so you can find and use the least famous features. Most of the examples and tutorials are available as a collection of Jupyter notebooks. The next chapter, Jupyter Notebooks Gallery, presents a list describing each notebook, what they will teach, what they can be used for, for which robot, etc.

All Poppy libraries are open source and are released under the GPL v3 license. So you can freely access the source code on github. Do not hesitate to fork them, send pull request and contribute!

Why Python and Anaconda?

The libraries developed for the Poppy project were designed with the aim to make it easy and fast to write code for controlling various robots based on - originally - robotis dynamixel motors. The idea was to provide access from the lower level - raw serial communication with a specific motor for instance - to higher levels such as starting and stopping primitives/behaviors (e.g. face tracking, postures, ...) or directly recording motions through learning by demonstration.

We decided to write most of them in Python as its flexibility allows for fast and modular development. It was also meant to be accessible by a large audience, from developers and roboticists in general, to hobbyists, researchers, artists... Python was also chosen for the tremendous pools of existing libraries (scientific, computer vision, IO, web...) so if one is interested in adding a new feature, such as support for a new motor/sensor, it should be as easy and fast as possible.

Finally, support for multiplatforms and ease of installation were also key aspects.

We also strongly advise to use the Anaconda Python distribution as it already includes most of the libraries needed by the Poppy libraries. We also provide all poppy libraries as conda recipes so they can be easily install using Anaconda (see the install section).

Overview of the different libraries

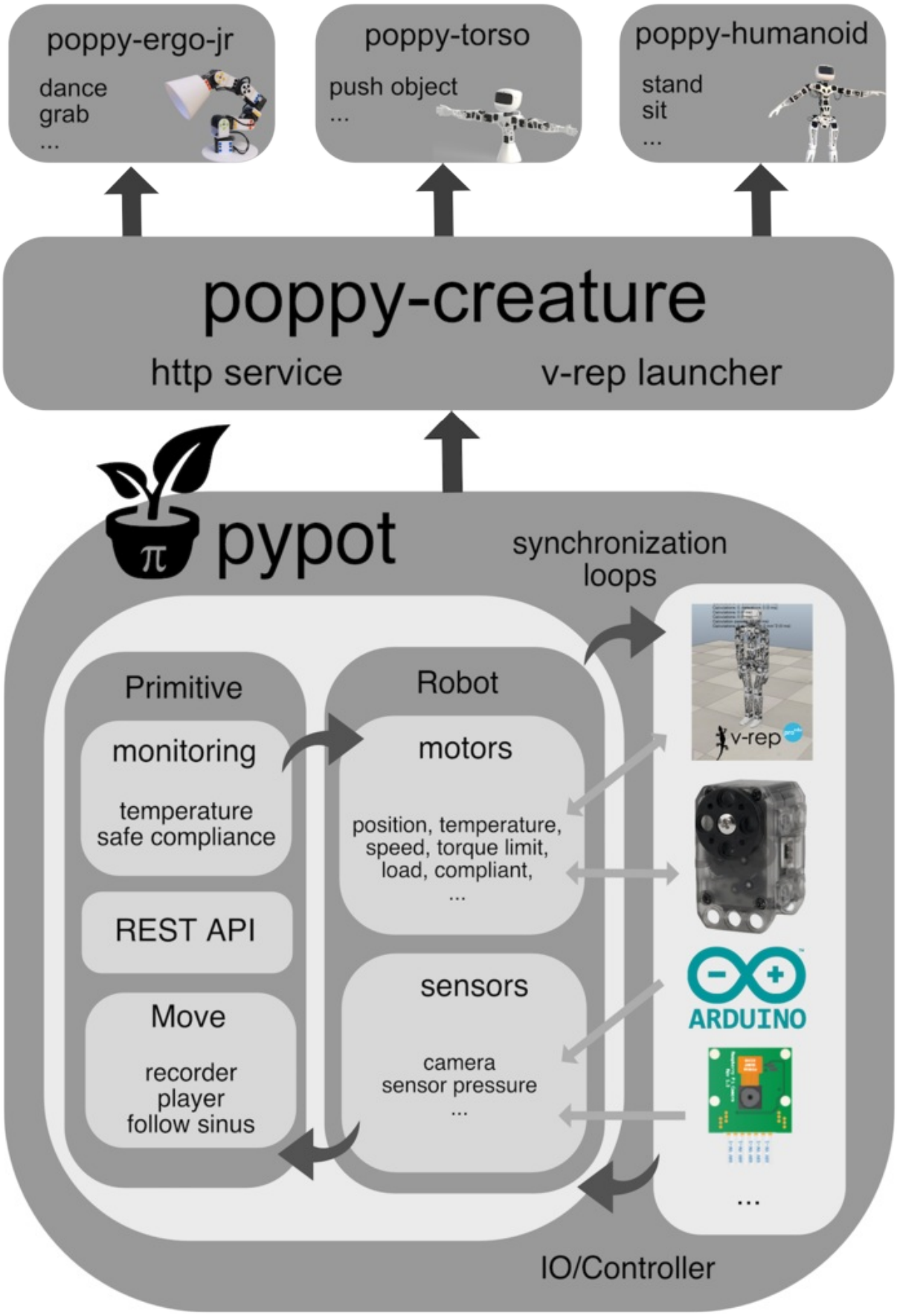
> A more detailled documentation of these software libraries is available in the software libraries section

Before jumping into the code, we will briefly introduce the different existing Poppy libraries and how they interact with each other.

They are three main library levels:

- pypot: This is the core of the Poppy software architecture. Pypot handles all the low level communication with the hardware (both sensors and motors), defines synchronization loops so your command are always up to date. It also provides the primitives mechanism which allows the definition of simple behavior that can be - more of less - automatically combined.
- poppy-creature: This library defines the common tools shared by all Poppy robots, for instance how to automatically launch the simulator or start the HTTP API attached to any robot.
- poppy-ergo-jr, poppy-torso, and poppy-humanoid: Those libraries are specific to their respective Poppy robot. They define the particular configuration of the robot, the sensors it uses, which motors are connected to which buses... This is also were behaviors specific to a creature are defined (the stand primitive for the humanoid for instance).

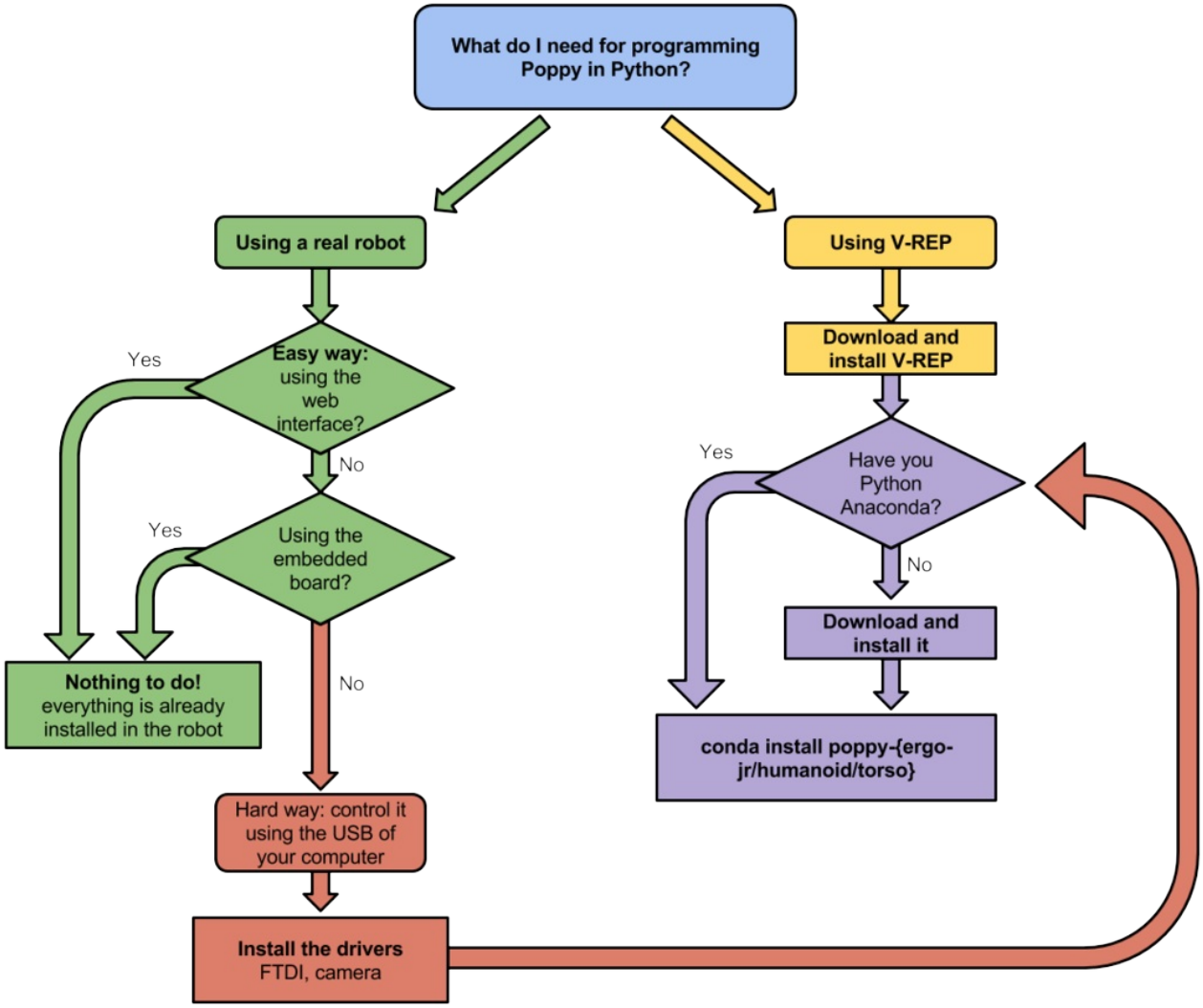
This is summarized in the diagram below:



Installation

First, note that if you are only planning to use real robots, they already come with Python and all Poppy libraries installed. You can directly connect to the Jupyter notebook server via the web interface and have nothing to install on your machine!

What you need to install is summarized in the diagram below:



Yet, if you are planning to either

- Use a simulator (e.g. V-REP, or web simulator),
- or want to directly plug the robot to your computer

You will have to install Poppy libraries locally. They work on Windows, Mac OSX, Linux, and have been tested on:

- Python >= 2.7
- Python >= 3.4

Also note that if you are planning to directly plug your robot to your USB port, specific drivers should be installed.

All steps are detailed in the chapter *install Poppy software*.

Quickstart: Hello Poppy world!

To give you a rapid overview of what you can do using Python to program Poppy robots, this section will show you how to:

- Create and connect your robot
- Retrieve values from the sensor and send motor commands
- Start playing with primitive by recording motions by demonstration

This section does not intend to cover everything that can be done in Python with Poppy but to give you sneak peaks of the most common features. For more advanced use, you should refer to the next section where we present a list of Jupyter notebooks each detailing a specific aspect or feature.

In the following examples, we assume that you have a working environment meaning that you either:

- are using the Python embedded in your robot: through the Jupyter Notebook server,
- or you have installed everything locally to work with a simulator.

Create and connect to a Poppy robot

Import the library

The very first step you have to do to start programming Poppy robots in Python is to import the library. In Python they are called module or package.

To do that, you write something similar to:

```
from poppy.creatures import *
```

This will actually import all Poppy robots installed on the Python distribution you are using. If you want to use a specific robot, you can replace the * (which means all here) by the name of the robot you want.

For the ErgoJr:

```
from poppy.creatures import PoppyErgoJr
```

For the Torso:

```
from poppy.creatures import PoppyTorso
```

For the Humanoid:

```
from poppy.creatures import PoppyHumanoid
```

> If you see an error similar to the one below when executing the previous line, this means that the libraries are not correctly installed. See the section *install Poppy software*.

```
In [1]: from poppy.creatures import PoppyHumanoid
ImportError Traceback (most recent call last)
<ipython-input-1-18e4c5a36525> in <module>()
----> 1 from poppy.creatures import PoppyHumanoid

ImportError: cannot import name PoppyHumanoid
```

Create the Robot object - with a real robot

Then, you can actually create the Python object that will represent your robot. Depending on the Poppy robot you are using:

```
# if you are using an Ergo Jr
poppy = PoppyErgoJr()
```

or

```
# if you are using a Torso
poppy = PoppyTorso()
```

or

```
# if you are using a Humanoid
poppy = PoppyHumanoid()
```

And that's it, if you did not see any error message it means that you are connected to your robot. If you see an exception like the one shown below, you should check the wire connection and try again:

```
IOError: Connection to the robot failed! No suitable port found for ids [3, 5, 7, 11, 13, 17]. These ids are missing [
3, 5, 7, 11, 13, 17] !
```

Create the Robot object - with V-REP

To use a simulated robot instead of a real one, you only have to specify it when creating the Robot object. For instance, if you want to create a simulated Poppy Torso, you simply have to execute the following line:

```
poppy = PoppyTorso(simulated='vrep')
```

All three Poppy robots - Humanoid, Torso, and Ergo Jr - can be used with V-REP.

If you see an error message like this, check that you have launched V-REP and that you have close the popup in V-REP (see this chapter for details).

```
IOError: Connection to V-REP failed!
```

Create the Robot object - with web simulator

Currently only the Ergo Jr is usable within the web simulator. It also requires specific versions of libraries to be used properly.

To make sure you meet these requirements, you can type this command from your shell:

```
pip install pypot>=2.12 poppy-creature>=1.8 poppy-ergo-jr>=1.6 --upgrade
```

You can then instantiate the poppy-ergo-jr creature:

```
poppy-services --poppy-simu --snap --no-browser poppy-ergo-jr
```

This will create a server for Snap*!* on port 6969, and a server for the visualizer on port 8080.

You can then head to the visualizer page.

Access the sensors and motors

The robot object you just created contains two main groups of objects:

- motors
- sensors

that can be easily access using *poppy.motors* and *poppy.sensors*. As soon as the robot object is created it automatically starts synchronization loops which will ensure that the last available value are received/sent to the robot.

> Servomotors that are used in Poppy robots can be seen as both motors and sensors. Indeed, on top of being "simple" motors, they also provide multiple sensing information: their current position, speed and load but also their temperature, the current used... Yet, for simplification they are only available under the motor category.

Get data from your robot

Now that you have created your robot object, you can directly use Python to discover which motors are attached.

> In all examples below the results are shown for an ErgoJr. If you are using a Torso or a Humanoid you will see more motors with different names.

For instance, to know how many motors your robot have you can execute:

```
print(len(poppy.motors))
```

poppy.motors is actually a list of all motors connected to your robot. Thus, if you want to get the present position of all motors, you can do:

```
for m in poppy.motors:
    print(m.present_position)
```

Of course, you can also access a specific motor. To do that, you need to know the name for the motor you want to access. You can find this list in the assembly documentation of your robot.

You can also get a list of all motors name directly from python:

```
for m in poppy.motors:
    print(m.name)
```

or using a motor pythonic expression:

```
print([m.name for m in poppy.motors])
```

Then you can directly access the desired motor by its name:

```
m = poppy.m3
```

or get its position:

```
print(poppy.m3.present_position)
```

The most common values for motors are:

- present_position
- present_speed
- present_load

Similarly, you can get data from your sensors. Depending on the Poppy robot you have different sensors available. You can get the list of all sensors in the exact same way you did for motors:

```
print([s.name for s in poppy.sensors])
```

And then access a specific sensors by its name. For instance, to get an image from the camera of the Ergo Jr:

```
img = poppy.camera.frame
```

> This section just presented some of the available values that you can get from your motors/sensors. They are many other - some are specific to a particular robot - we will present them through the different notebooks.

Send motor commands

Now that we have shown you how to read values from your robot, it is time to learn how to make it move!

This is actually really similar to what you have just seen. Instead of getting the *present_position* of a motor you simply have to set its *goal_position*.

But first, you have to make sure your motor is stiff, meaning that you cannot move it by hand. To do that we will turn off its compliancy. Assuming you have an Ergo Jr and want to make the motor *m3* moves - feel free to use any other motor but make sure the motor can freely move without hurting any of your finger:

```
poppy.m3.compliant = False
```

The motor should now be stiff. And then, to make it move to its zero position:

```
poppy.m3.goal_position = 0
```

> Note: *present_position* and *goal_position* are actually two different registers. The first refers to the current position of the motor (read only) while the second corresponds to the target position you want your robot to reach. Thus, they can have different values while the motor is still moving to reach its *goal_position*.

As a slightly more complex example we will make it go to 30 degrees then -30° three times:

```
import time

for _ in range(3):
    poppy.m3.goal_position = 30
    time.sleep(0.5)
    poppy.m3.goal_position = -30
    time.sleep(0.5)
```

Note that after each new value set to *goal_position* we wait so the motor has enough time to actually reach this new position. Another way to do the same thing is to use the goto_position method:

```
import time

for _ in range(3):
    poppy.m3.goto_position(30, 0.5, wait=True)
    poppy.m3.goto_position(-30, 0.5, wait=True)
```

As you can see, this method takes three arguments, the target position, the duration of the move and whether to wait or not the end of the motion.

If you want to move multiple motors at the same time, you can simply do something like:

```
for _ in range(3):
    poppy.m1.goal_position = -20
    poppy.m3.goal_position = 30
    time.sleep(0.5)
    poppy.m1.goal_position = 20
    poppy.m3.goal_position = -30
    time.sleep(0.5)
```

or use a python dictionary storing the target position per motor you want to move, that can be given to the goto_position method:

```
pos_1 = {'m1': -20, 'm3': 30}
pos_2 = {'m1': 20, 'm3': -30}

for _ in range(3):
    poppy.goto_position(pos_1, 0.5, wait=True)
    poppy.goto_position(pos_2, 0.5, wait=True)
```

You can turn a motor back to its compliant mode (where you can freely move it) by setting its compliant register to True:

```
poppy.m3.compliant = True
```

Record and play motion by demonstration using primitives

Pypot provides you with the primitive mechanism, which are simply pre-defined behaviors that can be attached to your robot. In this section, we will show you how to use some primitives already existing for recording and playing motions. You can also define your own primitive but this is out of the scope of this section, you will find details on how to do this in dedicated notebooks.

Record a motion by demonstration

Designing choreographies for your robot using *goal_position* or *goto_position* can be long and kind of troublesome. Fortunately, there is a much more efficient way of doing this: recording motions by directly demonstrating the move on the robot.

This can be summarized into few steps:

- make the robot compliant so you can move it by hand
- start the recording
- actually moves the robot so it follows whatever move/choreography you can think of
- stop the recording

And now to do that in Python:

So, first we turn all motors of the robot compliants:

```
for m in poppy.motors:
    m.compliant = True
```

You can also record a movement with motors stiff (`compliant = False`), and moving them with goal_position or goto_position commands.

Then, we have to include the primitive used for recording motion:

```
from pypot.primitive.move import MoveRecorder
```

To create this primitive, you have to give the following arguments:

- on which robot you want to use this primitive (this can be useful if you are working with multiple robot at a time - for instance you can record a move on a robot and at the same time make it reproduce by another one. * the record frequency of the move you want to register: how many position per second will be recorded - the higher the more accurate the record will be but also more data will have to be processed - good values are usually between 10Hz and 50Hz.
- the motors that you want to record. you can record a move on a subpart of you robot, for instance only on the left arm.

Here, we will record a move on the whole robot at 50Hz:

```
recorder = MoveRecorder(poppy, 50, poppy.motors)
```

> We used *poppy.motors* to specify that we want all motors if you only want let's say the two first motors of an Ergo Jr you could have used *[poppy.m1, poppy.m2]* instead.

Now it is time to record. As it can be hard to both move the robot and type Python command at the same time, we will make a small script, that:

- wait 5s so you can get ready to record
- start the record
- record for 10 seconds
- stop the records

```
import time

# Give you time to get ready
print('Get ready to record a move...')
time.sleep(5)

# Start the record
record.start()
print('Now recording !')

# Wait for 10s so you can record what you want
time.sleep(10)

# Stop the record
print('The record is over!')
record.stop()
```

Now, you should have a move recorded. You can retrieve it from the recorder primitive:

```
my_recorded_move = record.move
```

and check how many positions where recorded:

```
print(len(my_recorded_move.positions()))
```

Replay recorded moves

Now to play back recorded motions you have to use another primitive: MovePlayer

```
from pypot.primitive.move import MovePlayer

player = MovePlayer(poppy, my_recorded_move)
```

As you can see, to create it you have to specify the robot (as for the MoveRecorder) and the move you want to play.

> Automatically all recorded motors become stiff to be able to play the move.

Then, you can simply start the replay:

```
player.start()
```

And if you want to play it three times in a row:

```
for _ in range(3):
    player.start()
    player.wait_to_stop()
```

We use the *wait_to_stop* method to make sure we wait for the first move to finish before we start another. By default, playing a move we will not block to allow you to play multiple move in parallel.

Write a simple sensori-motor loop

Robotic is all about sensori-motor loops, meaning that motor commands will be more or less directly related to the sensor readings. In other terms the robot actions will be determined by what it perceives from its environment.

Poppy libraries and more particularly pypot provides you with tools to easily write sensori-motor loops. We will show here a very simple example where some motor of an Ergo Jr will be controlled by the position of other motors in order to keep the head of the Ergo Jr straight.

To do that, we will free the two first motors, so they can be moved by hand. Two other motors will try to lively compensate the motion applied on the free motors.

We need few simple steps:

1. read values from sensors (here the two free motors)
2. compute command from those readings
3. set new motor command
4. go back to step 1.

> This example is designed for the Ergo Jr. It could be adapted to other Poppy robots, by changing the motors used. Yet, it is not that obvious which one to use to have a "cool" result.

Demo version

Before writing the sensori-motor loop, we will first set the Ergo Jr in a base position.

```
from poppy.creatures import PoppyErgoJr

jr = PoppyErgoJr()

jr.goto_position({'m1': 0.,
                  'm2': -60.,
                  'm3': 55.,
                  'm4': 0.,
                  'm5': -55.,
                  'm6': 60.}, 2., wait=True)
```

Then, we make sure the *moving speed* of the motors are not too high to prevent shaky motions:

```
for m in jr.motors:
    m.moving_speed = 250
```

Finally, we free the two first motors:

```
jr.m1.compliant = True
jr.m2.compliant = True
```

Now, that everything is setup we write our very simple sensori-motor loop like this:

```
import time

while True:
    # Step 1
    p1 = jr.m1.present_position
    p2 = jr.m2.present_position

    # Step 2
    g1 = -p1
    g2 = -p2

    # Step 3
    jr.m4.goal_position = g1
    jr.m6.goal_position = g2

    time.sleep(.02)
```

- **Step 1:** As you can see, here our readings step is simply to retrieve the *present_position* of the motors *m1* and *m2*.
- **Step 2:** Here, we defined the base position so the motors *m1*/*m4* and *m2*/*m6* are parallel. Thus, to compensate the head position, we simply have to define the new motor goal position as the opposite of the read present position.
- **Step 3:** We simply set the goal position as the just computed command

Those steps are included inside an infinite loop - with a `time.sleep` to avoid CPU overhead.

> To stop this *while True* loop, you will have to use the classical Ctrl-c, or use the stop button if you are running it through Jupyter.

Now with a primitive

But what about if you want to make this behavior an independent "brick" that you can start/stop on demand combine with other behaviors. Well, primitives are meant to do just that.

There is two main types of primitive: *Primitive* and *LoopPrimitive*. The first one basically gives you access to just a *run* method where you can do everything you want on a robot. The second one as the name indicates is an infinite loop which calls an *update* method at a pre-defined frequency. In our case it is the more suited one.

Here is the entire definition of this primitive:

```
class KeepYourHeadStraight(LoopPrimitive):
    def setup(self):
        for m in self.robot.motors:
            m.compliant = False

        self.robot.goto_position({'m1': 0.,
                                  'm2': -60.,
                                  'm3': 55.,
                                  'm4': 0.,
                                  'm5': -55.,
                                  'm6': 60.}, 2., wait=True)

        for m in self.robot.motors:
            m.moving_speed = 250

        self.robot.m1.compliant = True
        self.robot.m2.compliant = True

    def update(self):
        self.robot.m4.goal_position = -self.robot.m1.present_position
        self.robot.m6.goal_position = -self.robot.m2.present_position
```

As you can see, there is two main parts. The ***setup*** method which defines what needs to be done to prepare the robot before starting the behavior - here simply puts it in its base position and turn on the compliance for the two first motors.

And the ***update*** method which will be regularly called: here is where we put the actual code for the sensori-motor loop: reading sensor - computing the new command - and sending the new command to the motors.

Now that we have defined our primitive, we can instantiate it and start it:

```
# we specify we want the primitive to apply on the jr robot instance
# and that the update method should be called at 50Hz
head_straight = KeepYourHeadStraight(jr, 50.0)

head_straight.start()
```

You can stop it whenever you want:

```
head_straight.stop()
```

And re-starting it again...

```
head_straight.start()
```

The huge advantage of using a primitive in this case is that after starting it, you can still easily run any other codes that you want. The primitive starts its own thread and thus runs in background without blocking the execution of the rest of the code.

Use the REST API to control a Poppy Robot

> This page is not currently written. Your help is welcome to fulfill it !

Gallery of activities

> This section is not currently written. Your help is welcome to fulfill it !

Contrôler Poppy avec un Arduino via Snap4Arduino

Written by Gilles Lassus.

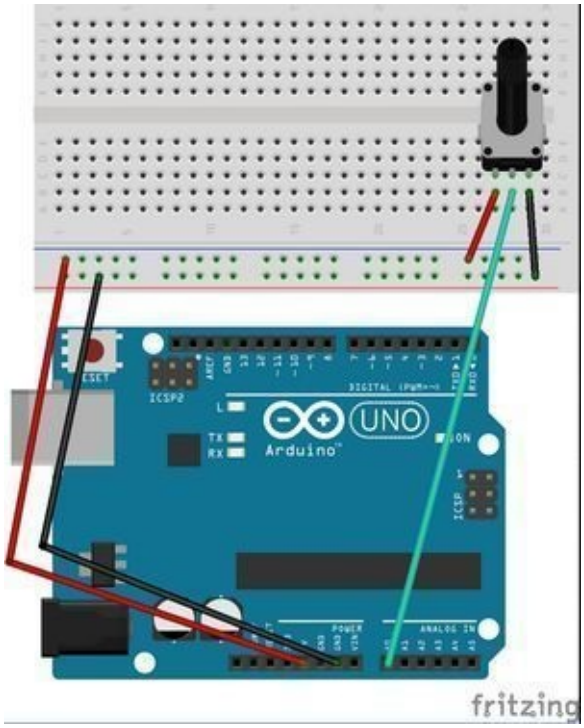
Objectif : contrôler un moteur de Poppy par un potentiomètre sur la platine Arduino.

Préparation de Snap4Arduino

- Téléchargez et installez Snap4Arduino.
- Téléchargez les blocs pypot-snap-blocks.xml. Ils devront être importés dans Snap4Arduino à chaque démarrage.

Préparation de l'Arduino

- Connectez votre platine, ouvrez Arduino et téléversez le firmware StandardFirmata. (disponible via Fichier - Exemples - Firmata).
- Branchez un potentiomètre sur la sortie analogique A0, comme illustré ci-dessous :



Lancement de la simulation (dans le cas d'un Poppy simulé dans Vrep)

- Lancez Vrep .
- Exécutez les commandes python suivantes :

```
from poppy.creatures import PoppyHumanoid

poppy = PoppyHumanoid(simulator='vrep', use_snap=True)
```

puis

```
poppy.snap.run()
```

Ouverture de Snap4Arduino

- Lancez Snap4Arduino et importez les blocs *pypot-snap-blocks.xml*. (une fenêtre avertissant que le projet importé a été créé par

Snap*!* apparaît ; elle est sans conséquence.)

- Dans les blocs Arduino, cliquez sur *Connect Arduino* pour établir la connexion entre Snap4Arduino et votre platine.

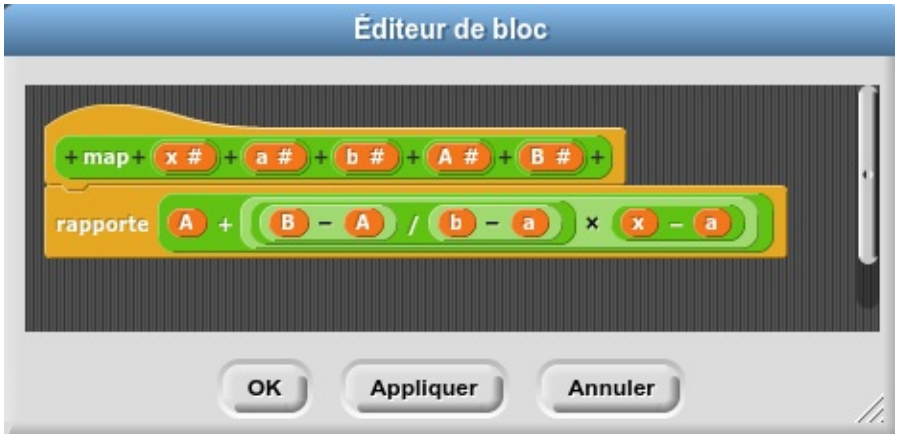
Un message de confirmation apparaît, signe que la connexion est effective.

Commander un moteur via le potentiomètre

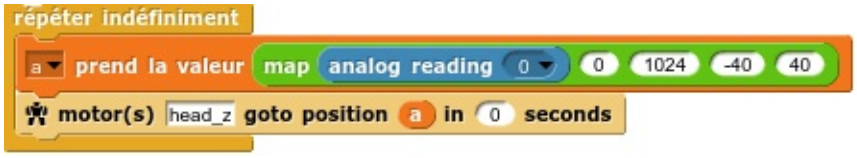
La valeur analogique lue dans A0 est un entier entre 0 et 1024. Pour la "mapper" entre (environ) -40 et 40, on la divise par 12 avant de lui soustraire 40. On peut donc alors construire l'instruction suivante, qui fera bouger le moteur *head_z* de Poppy entre -40° et +40° :

Remarques diverses

- Il peut être utile de créer un bloc *map* équivalent à la fonction éponyme d'Arduino, permettant de mettre à l'échelle automatiquement une valeur dans une plage donnée :



Le script de commande du moteur *head_z* de Poppy entre -40° et 40° deviendrait alors :



Cette méthode de contrôle a pour principal défaut de "bloquer" la carte Arduino avec le StandardFirmata : il serait plus agréable de pouvoir simplement lire les données du port série envoyées par l'Arduino, et ainsi pouvoir téléverser le programme de son choix dans l'Arduino. Ceci est discuté ici. Toutefois, la page du projet Snap4Arduino liste les composants annexes (LCD display, UltraSound Sensor) pouvant être directement contrôlés, et explique en détail comment modifier le StandardFirmata pour intégrer un nouveau composant.

Switching from a simulated to a real Poppy robot

A key feature of the Poppy project is to let you, as seamlessly as possible, switch from a simulated robot (e.g. using V-REP) to a real one. It is particularly useful when:

- Developing an experiment where you can setup everything using the simulation, then run it on the real robot.
- Working in a classroom context where students can work on their own computer via the simulation and share a robot for real world tests

While it has been designed to cut the effort needed to switch from one to the other, there are still a few steps to understand. One of the major difference is when you are working in simulation everything runs on your own computer while when you are using a real robot, the software (e.g. the Python Notebooks) actually runs in the robot.

Using Snap*!*

> This page is not currently written. Your help is welcome to fulfill it !

Using Jupyter Python Notebooks

One of the advantages of working with Jupyter Notebooks is the possibility to use it as a client/server approach.

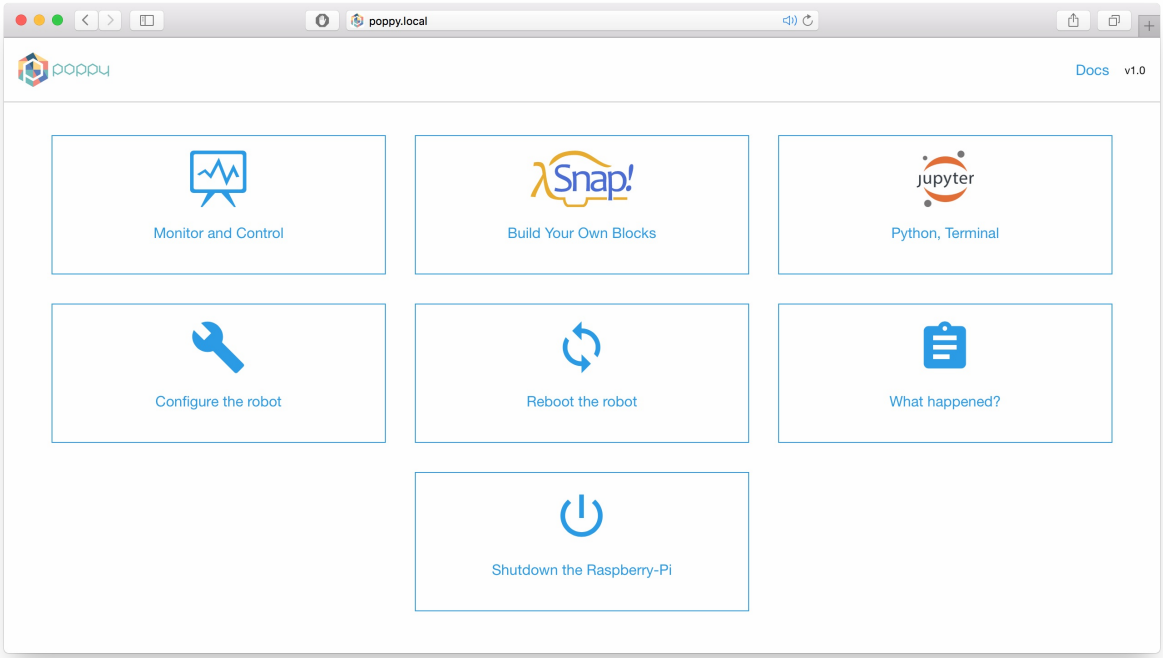
Each Poppy robot hosts a Jupyter server accessible via the web interface (see section quickstart for details).

When working in simulation, everything is run and stored locally on your computer. When working with a real robot you can program it from a web browser on your own machine but your notebooks are actually stored and run in the robot.

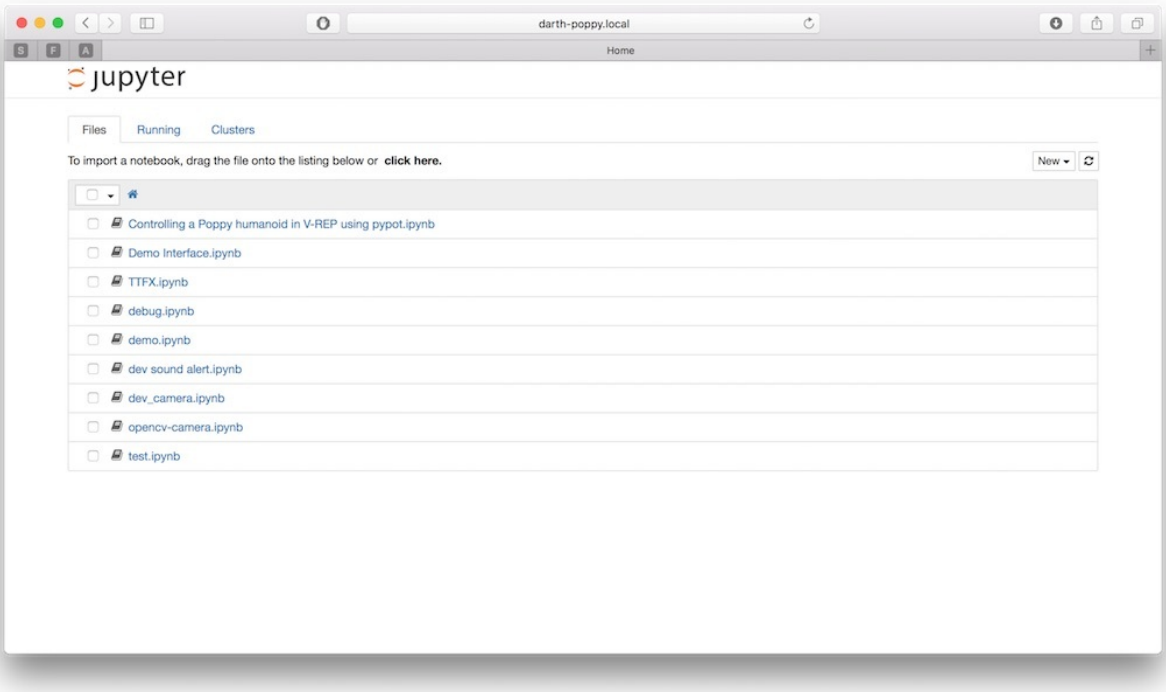
Thus to switch from simulation to a real robot, you need to switch from your local instance of Jupyter to the remote one hosted by the robot. The steps are described below.

Connect to the Jupyter on the robot

Once connected to the robot web interface *http://poppy.local* (we will assume here its hostname is *poppy*, just replace it by the new hostname if you changed it), you should see a **open Ipython notebook** link.



When clicked it will start Jupyter on the robot and redirect you to the Jupyter webserver. You should then see the root of the notebook folder hosted on the robot:

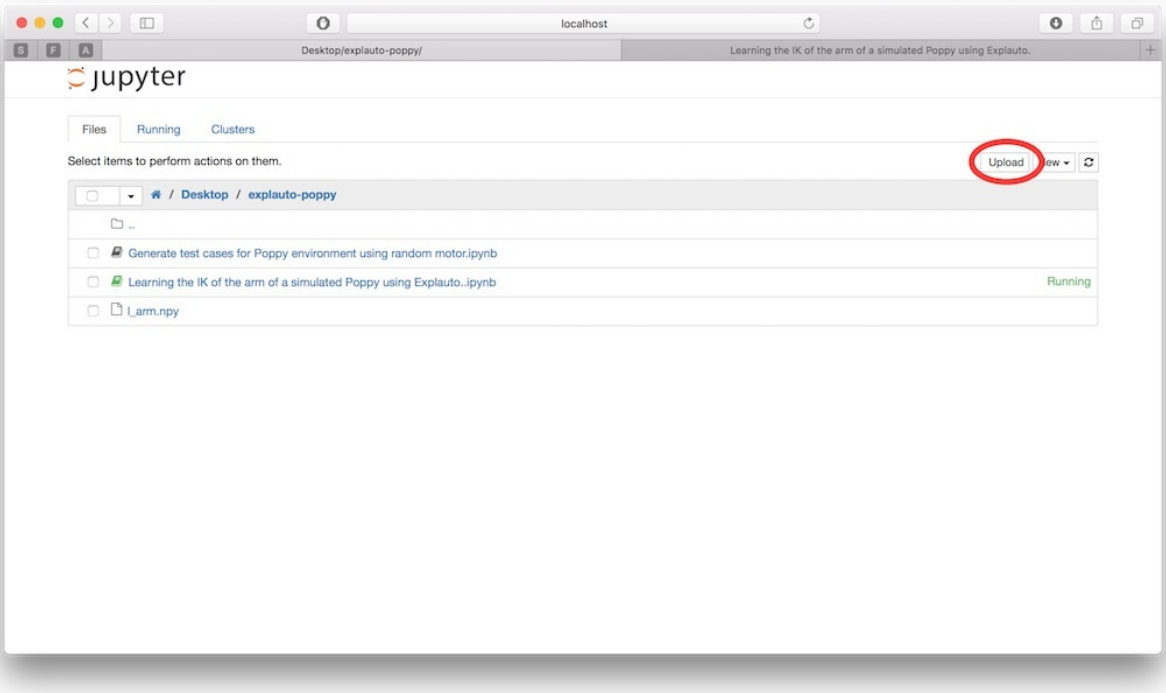


This is where you can put your own notebooks. Of course, you can create folder, organize them as you want, etc...

Note: If you need finer access or more advanced configuration (such as permission for instance), you have to log directly to the robot using ssh.

Upload a notebook

Once connected to the Jupyter server hosted by the robot, you can directly use the Jupyter interface for uploading new notebooks.



The circled button let you *upload* your local notebook, so stored on your own machine, to the robot. They can then be directly run on the robot.

Be aware that at the moment, we do not deal with sessions or permissions (as JupyterHub does for instance), and thus anyone with access to the robot can use or delete all notebooks stored in the robot.

Adapt your code

There is few places where you should actually modify your code so it works with a real robot. We try to minimize the effort needed as much as possible, yet some steps are still required.

Instantiation

When creating the robot, you actually need to specify if you are willing to work with a real or a simulated robot. This is simply done via a parameter. For instance:

When working with V-REP:

```
from poppy.creatures import PoppyHumanoid

poppy = PoppyHumanoid(simulator='vrep')
```

Will become for a real robot:

```
from poppy.creatures import PoppyHumanoid

poppy = PoppyHumanoid()
```

Of course, this works for all existing Poppy creatures: Humanoid, Torso and ErgoJr.

This is most of the changes that you should do.

Specific APIs

Some part of the API are platform specific. For instance, when using V-REP you have access to *tracking* features that let you retrieve any object 3D position. Of course, such method do not have a real world equivalent and thus are not available when working with a real robot.

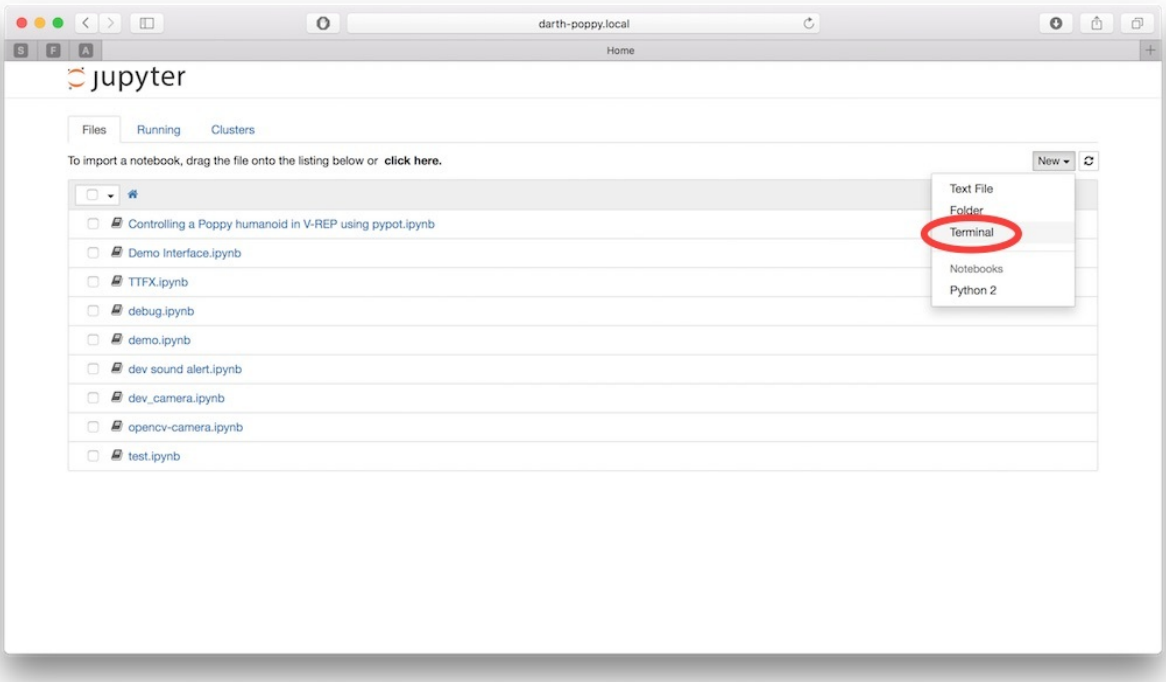
A good practice if you want to write code compatible for both cases is to use the *simulated* property. It is automatically set to the correct value depending on how your instantiate your robot. For instance,

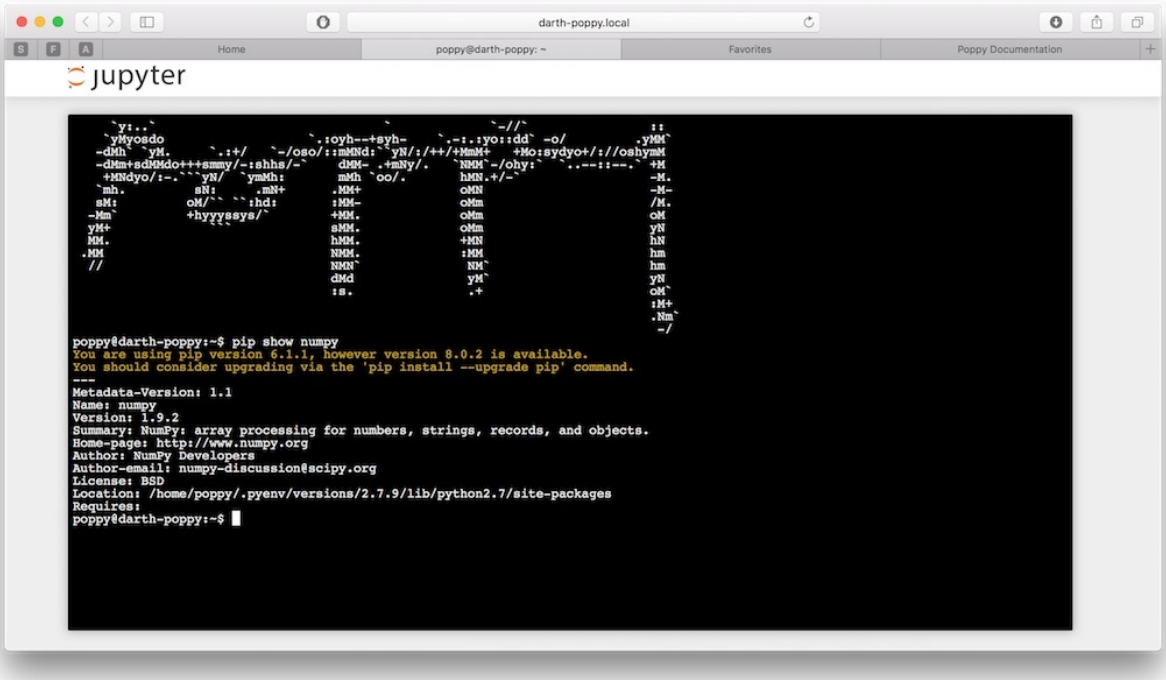
```
poppy = PoppyHumanoid(simulator='vrep')

def reset_position():
    if poppy.simulated:
        poppy.reset_simulation()
    else:
        print('Ask one internship student to put the robot back in its origin position.')
        time.sleep(10)
```

Version and 3rd party libraries

The main drawback of this client/server way of working is that your locally installed software versions may differ from the one installed on the robot.

The Python installed on the robot is Python 2.7 and comes with most of the scientific main libraries (numpy, scipy, matplotlib, opencv). An exhaustive list of the installed Python packages will be available soon . At the moment, the easier way to get it is to used a *terminal notebook* which can be directly run from the Jupyter interface.





Using the same technique, you can install 3rd party libraries directly on the robot. The pip and conda utility tools are installed and should be used when possible.

_Note that the embedded board are based on armv7 and thus some libraries may be hard to compile. We are maintaining a list of conda recipes specifically built for this platform here.

Contributions are more than welcomed! _

Software libraries

This section will provide software documentation of various libraries used in Poppy robots.

- Pypot
- Poppy-creature
- Poppy Ergo Jr
- Poppy Humanoid
- Poppy Torso

Pypot library

> Pypot documentation has still not been merged in the new documentation. You can find it at poppy-project.github.io/pypot/

Poppy-creature library

Introduction

Poppy-creature is a small library providing an abstract interface for robots (Poppy Humanoid, Poppy Torso, Poppy Ergo Jr...). It links high level controls and pypot, the generic low level library.

It mainly contains the class definition of poppy.creatures.abstractcreature.AbstractPoppyCreature which takes a configuration and builds a pypot.robot.robot.Robot out of it, but also a bunch of parameters to launch Snap! or HTTP servers, or to replace the communication toward Dynamixel servos by a communication with a simulator.

The arguments you can provide are:

- `base_path` default: None Path where the creature sources are. The librarie looks in the default PATH if not set.
- `config` default: None Path to the configuration file with respect to the base-path
- `simulator` default: None Possible values : 'vrep' or 'poppy-simu'. Defines if we are using a simulator (and which one) or a real robot.
- `scene` default: None Path to the scene to load in the simulator. Only if simulator is vrep. Defaults to the scene present in the creature library if any (e.g. poppy_humanoid.ttt).
- `host` default: 'localhost' Hostname of the machine where the simulator runs. Only if simulator is not None.
- `port` default: 19997 Port of the simulator. Only if simulator is not None.
- `use_snap` default: False Should we launch the Snap! server
- `snap_host` default: 0.0.0.0 Hostname of the Snap! server
- `snap_port` default: 6969 Port of the Snap! server
- `snap_quiet` default: True Should Snap! not output logs
- `use_http` default: False Should we launch the HTTP server (for
- `http_host` default: 0.0.0.0 Hostname of the HTTP server
- `http_port` default: 8080 Port of the HTTP server
- `http_quiet` default: True Should HTTP not output logs
- `use_remote` default: False Should we launch the Remote Robot server
- `remote_host` default: 0.0.0.0 Hostname of the Remote Robot server
- `remote_port` default: 4242 Port of the Remote Robot server
- `sync` default: True Should we launch the synchronization loop for motor communication

The sources are available on GitHub.

Poppy services

Poppy-creature also provides a command line utility *poppy-services*. It provides shortcuts to start services like SnapRemoteServer and HTTPRemoteServer from your terminal. Example:

```
poppy-services poppy-ergo-jr --snap --no-browser
```

This will launch the SnapRemoteServer for a real Poppy Ergo Jr robot.

> The `--no-browser` option avoid the automatic redirection to the *Snap!* webpage. You can remove it if you use a computer with a GUI (e.g your laptop instead of the robot embedded board).

Another example:

```
poppy-services poppy-ergo-jr --snap --poppy-simu
```

It will open a *Snap*!__ windows for a simulated poppy-ergo-jr.

The way to use it is:

```
poppy-services <creature_name> <options>
```

the available options are:

- `--vrep` : creates the specified creature for using with V-REP simulator
- `--poppy-simu` : creates the specified creature for using with web simulator and also launches the HTTP server needed by poppy-simu. Poppy-simu is only available for poppy-erg-jr for now.
- `--snap` : launches the Snap! server and directly imports the specific Poppy blocks.
- `-nb` or `--no-browser` : avoid automatic start of Snap! in web browser, use only with --snap
- `--http` : start a http robot server
- `--remote` : start a remote robot server
- `-v` or `--verbose` : start services in verbose mode (more logs)

Create your own Poppy creature

While developping a new Poppy creature, it is first easier to simply define it in a configuration file or dictionnary and instanciate a pypot.robot.robot.Robot from Pypot directly.

But when you want to make it easily usable and available to non-geek public, the best is to create your own creature's library. It should contain a configuration file and a class that extends poppy.creatures.abstractcreature.AbstractPoppyCreature. You can then add your own properties and primitives.

Example from Poppy Humanoid:

```
class PoppyHumanoid(AbstractPoppyCreature):
    @classmethod
    def setup(cls, robot):
        robot._primitive_manager._filter = partial(numpy.sum, axis=0)

        for m in robot.motors:
            m.goto_behavior = 'minjerk'

        for m in robot.torso:
            m.compliant_behavior = 'safe'

        # Attach default primitives:
        # basic primitives:
        robot.attach_primitive(StandPosition(robot), 'stand_position')
        robot.attach_primitive(SitPosition(robot), 'sit_position')

        # Safe primitives:
        robot.attach_primitive(LimitTorque(robot), 'limit_torque')
```

Package your code it properly using setuptools.

For a better integration with the Poppy installer scripts, please have in the root of your repo a folder named *software* containing:

- the installation files (setup.py, MANIFEST, LICENCE)
- a folder named poppy_yourcreaturename containing your actual code

At the end, don't forget to share it to the community! Most interesting creatures will be added to this documentation!

Installing

> poppy-creature library is a dependancy of any Poppy robots libraries, so you don't have to install it by hand in a normal case.

To install the poppy-creature library, you can use pip:

```
pip install poppy-creature
```

Then you can update it with:

```
pip install --upgrade poppy-creature
```

If you prefer to work from the sources (latest but possibly unstable releases), you can clone them from GitHub and install them with (in the software folder):

```
python setup.py install
```

Poppy-ergo-jr library

> This section need to be completed, contribution are welcome !

Poppy-humanoid library

> This section need to be completed, contribution are welcome !

Poppy-torso library

> This section need to be completed, contribution are welcome !

Appendix

Network

This section will contain a collection of tips for avoiding network troubles.

Where is Poppy ?

If you are looking for your robot on your network, look at the zeroconf chapter.

Getting involved in the Poppy project

If you want to take part of this project, maybe the first step is to become a member of the community on the Poppy forum. The forum is the very central place to exchange with users and contributors. You can freely come and talk about your project or ideas with your prefered language.

There are many ways to contribute to this project as the Poppy project involves a very large scope of disciplines:

- Engineering fields such as AI, computer science, mechanics, electronics, machine learning...
- Humanities such as cognitive science, psychology...
- Life science such as biology, biomechanics,...
- Community management, scientific mediation, communication...
- Design such as web design, object design, UX,...
- Art with the need of animator to create the illusion of life and emotions.

If you have no idea how you could help but you would want to, you are very welcome and you can **take a look at open issues on our GitHub and call for contributions**.

For github ninja, you can of course create issues to notify a problem or develop new amazing features and open pull requests to integrate your idea.

FAQ

> This Page is not currently written. Your help is welcome to fulfill it !

If you have any question please fill free to ask on the Poppy forum.

General issues

> Motors seems to be tired (low torque, jerky, ...), what can I do ?

Change your wires !