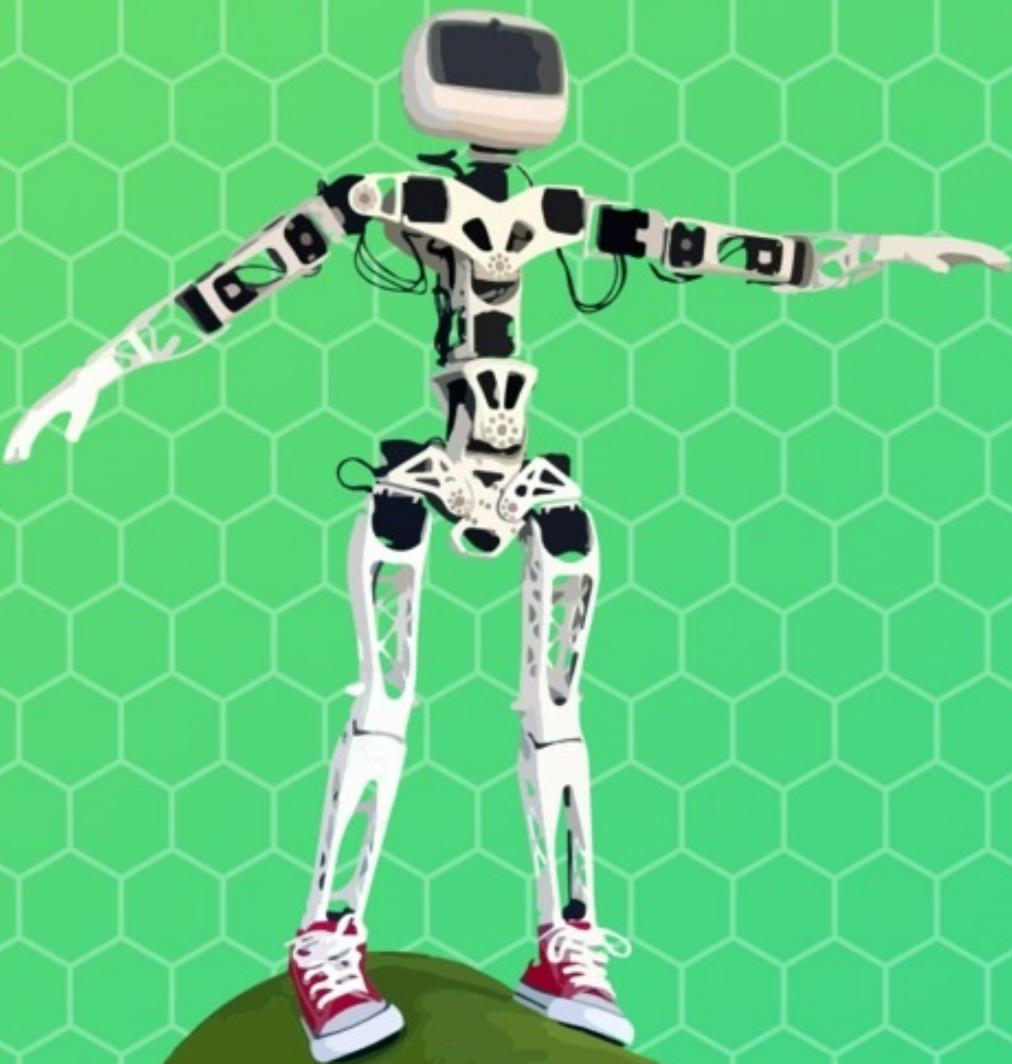


Poppy

Documentation

release 1.1.0



PyPot

Table des matières

Premiers pas

À propos de cette documentation	1.1
Mise en route	1.2

Installation des logiciels

Installation	2.1
Installez Bonjour/Zeroconf	2.1.1
Télécharger et graver l'image système	2.1.2
Installer les logiciels Poppy	2.1.3
Installation du simulateur V-REP	2.1.4
Installer les pilotes USB vers port série	2.1.5
Installer un ordinateur embarqué pour un robot Poppy	2.1.6

Assemblage du robot

Assembler l'Ergo Jr	3.1
Assemblage électronique	3.1.1
Configuration des moteurs	3.1.2
Assemblage des pièces mécaniques	3.1.3
Assembler Poppy Humanoid	3.2
Assembler Poppy Torso	3.3

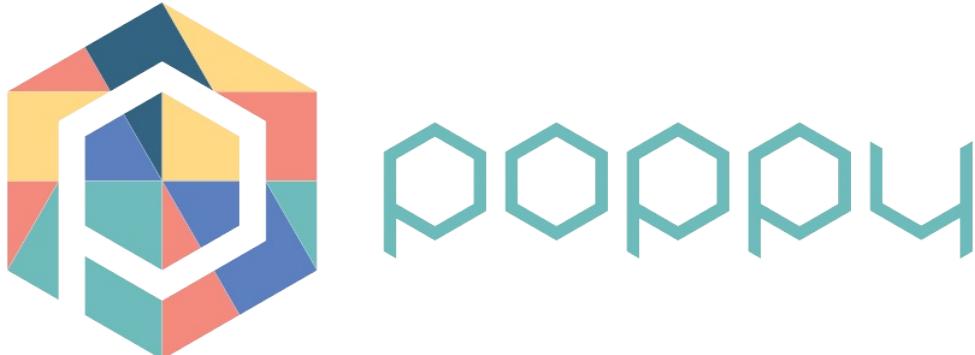
Programmation du robot

Programmation	4.1
Programmation avec Snap!	4.1.1
Utilisation des notebooks Jupyter	4.1.2
Programmation en Python	4.1.3
API des robots	4.1.4
Activités	4.1.5
Lien avec Snap4Arduino	4.1.6
De la simulation au robot tangible	4.2
Programmer avec les notebooks Jupyter	4.2.1
Snap! sur le robot tangible	4.2.2
Documentation des bibliothèques logicielles	4.3
Pypot	4.3.1

Poppy-creature	4.3.2
Poppy Ergo Jr	4.3.3
Poppy Humanoid	4.3.4
Poppy Torso	4.3.5

Annexe

Réseau	5.1
Contribuez	5.2
FAQ	5.3



Documentation du projet Poppy

Bienvenue

Bienvenue dans la documentation du [Projet Poppy](#), une plate-forme de robotique open-source.

Dans cette documentation, nous allons essayer de tout couvrir, en partant d'un bref aperçu de ce qui est possible avec les outils du projet, et en passant par tous les détails nécessaires pour construire un robot Poppy ou de reproduire une des activités pédagogiques.

Introduction

Dans le [premier chapitre](#) nous vous donnerons un aperçu simple mais exhaustif de ce que vous pouvez faire dans ce projet, ainsi vous pourrez rapidement vous concentrer sur les chapitres suivants qui couvrent les points qui vous intéressent. Tandis que certains chapitres avancés peuvent nécessiter des connaissances en mécanique, électronique ou informatique, la section [Mise en route](#), est destinée à être facilement accessible par tous les lecteurs.

À propos et contributions

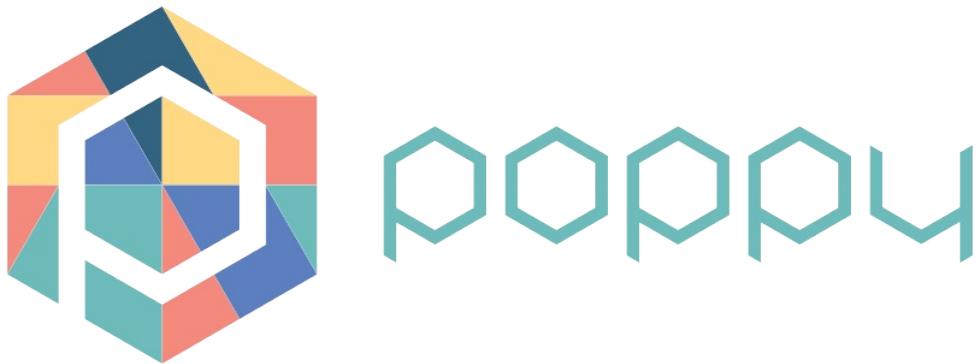
Cette documentation est maintenue par l'équipe Poppy avec l'aide de nombreux [contributeurs bénévoles](#). Si vous trouvez des erreurs ou que vous souhaitez mettre à jour le document veuillez suivre le [guide de contribution](#).

Ce document est sous [licence CC-BY 4.0](#). **Vous êtes libre de copier, modifier et redistribuer ce document tant que vous donnez un crédit approprié aux auteurs et un lien vers le site original [docs.poppy-Project.org](#).** Il est écrit en markdown, hébergé sur [GitHub](#), et [GitBook](#) est utilisé pour générer le site Web.

Version

Ce document a été mis à jour le Mo October yy.

Premiers pas



Le projet de [Poppy](#) est une plate-forme open-source pour la création, l'utilisation et le partage de robots interactifs imprimés en 3D. Il rassemble une communauté interdisciplinaire de débutants, d'experts, de chercheurs, d'enseignants, de développeurs et d'artistes. Ils partagent tous une même vision : les robots sont de puissants outils pour apprendre et développer la créativité, et ils collaborent pour améliorer le projet. Ils développent de nouveaux comportements pour les robots, créent des contenus pédagogiques, élaborent des performances artistiques, améliorent le logiciel ou même conçoivent de nouveaux robots.

La communauté [Poppy](#) développe des créations robotiques qui sont faciles à construire, personnaliser et utiliser.

Un ensemble de services web permet à la communauté de partager leurs expériences et de contribuer à l'amélioration de la plateforme Poppy.

Pour faciliter ces échanges, deux supports sont disponibles :

- [Le forum de projet Poppy](#) pour avoir de l'aide, partager ses idées et en discuter.
- [GitHub](#) pour soumettre vos contributions et faire un suivi des bugs logiciels.

Toutes les sources du projet Poppy (logiciel et matériel) sont disponibles sur [GitHub](#).

Le projet Poppy a été conçu initialement à [Inria dans l'équipe Flowers](#).

Les robots Poppy

Les robots Poppy sont open source et libre. Leur sources sont disponible librement, il est possible de les modifier et de les redistribuer selon les termes prévus par leur licenses. Les pièces matérielles (carte électronique et modélisation 3D) sont sous license ([Creative Commons Attribution-ShareAlike](#) et le logiciel [GPLv3](#)). Ils ont été tous conçus selon les même principes.

Les robots Poppy:

- sont fabriqués à partir de pièces imprimable en 3D et des servomoteurs Dynamixel,
- utilisent un ordinateur embarqué (Raspberry Pi ou Odroid pour les anciennes versions),
- fonctionnent avec une bibliothèque logicielle en Python, [pypot](#), qui permet de contrôler les servomoteurs Dynamixel simplement,
- sont également contrôlable avec les même outils dans un simulateur ([V-REP](#)),

- peuvent être contrôlé à l'aide d'un langage de programmation visuel ([Snap !](#) une variante de Scratch) et un langage textuel [Python](#). Ils sont également programmables via une API REST, ce qui permet de s'interfacer avec d'autres langages de programmation
- viennent avec une documentation associée, des tutoriels, des exemples et des activités pédagogiques.

Ils peuvent être utilisés comme tels ou hackés (au sens de bidouillé) pour explorer de nouvelles formes, ajouter des capteurs, etc...

Pour obtenir votre propre robot Poppy, vous pouvez soit : *vous procurer toutes les pièces en suivant la liste du matériel (voir ci-dessous)*. Acheter un des robots Poppy en kit complet chez notre [revendeur officiel](#), Génération Robots.

Poppy Ergo Jr

Le robot Poppy Ergo Jr est un petit bras robot à faible coût avec doté de 6 articulations. Il est fait de 6 moteurs peu cher (XL-320 servomoteurs Dynamixel servos) avec des pièces simples imprimées 3D.

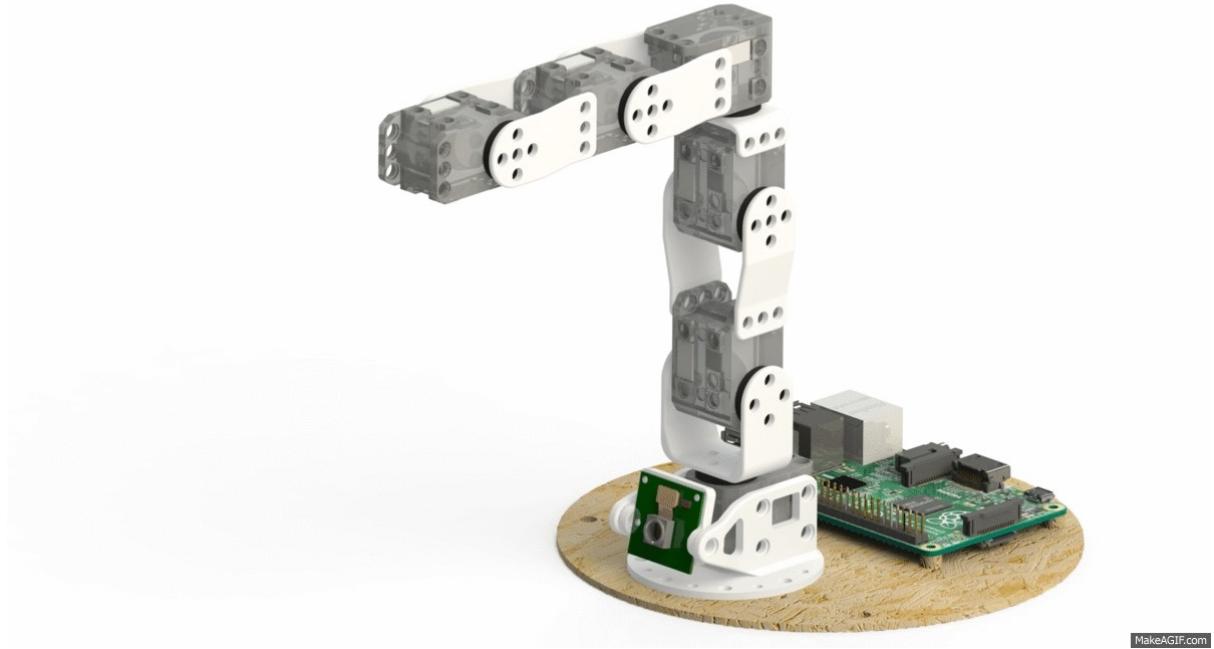


Les pièces 3D ont été conçues pour être facilement imprimable sur une imprimante 3D basique. Les moteurs coûtent seulement 20€ chacun. L'accès à sa carte électronique est simple. Elle facilite la connexion des capteurs supplémentaires et est bien adaptée à des fins pédagogiques.

Vous pouvez choisir parmi trois outils à la fin de son bras :

- Un abat-jour.
- Une pince.
- Un porte stylo.

Les rivets utilisés rendent le changement d'outil simple à faire. Vous pouvez l'adapter selon le type d'activités que vous faites.



MakeAGIF.com

Poppy Ergo Jr est le robot idéal pour apprendre la robotique sans difficultés : assemblage simple, facile à contrôler et à un prix abordable.

Vous pouvez vous procurez toutes les pièces en suivant [la nomenclature \(BOM\)](#) et imprimer les [pièces 3D](#) disponible au format STL.

Pour plus d'informations, jetez un coup d'oeil au [guide d'assemblage de l'Ergo Jr.](#)

Poppy Humanoid

C'est un robot humanoïde composé de 25 articulations avec une colonne vertébrale entièrement actionnée. Il est utilisé pour l'éducation, la recherche (étude de la marche, l'interaction Homme-robot) ou encore pour des performances artistiques. D'un bras au robot complet, cette plateforme est activement utilisée dans les laboratoires, les écoles d'ingénieur, les fablabs et les projets artistiques.

Vous pouvez vous procurez toutes les pièces en suivant [la nomenclature \(BOM\)](#) et imprimer les [pièces 3D](#) disponible au format STL.



Poppy Torso

C'est la partie supérieure du robot Poppy Humanoid (13 articulations). Poppy Torso est donc plus abordable qu'un Poppy Humanoid. Ce robot offre une solution plus adaptée aux besoins de l'éducation, des associations et des bricoleurs. Poppy Torso peut être un bon moyen pour apprendre les sciences, les technologies de l'information, l'ingénierie et les mathématiques (STIM).

Vous pouvez vous procurer toutes les pièces en suivant [la nomenclature](#). Les [modèles 3D](#) pour les pièces sont les mêmes que pour Poppy Humanoid, mais sans les jambes et avec un [support ventouse](#) en plus.

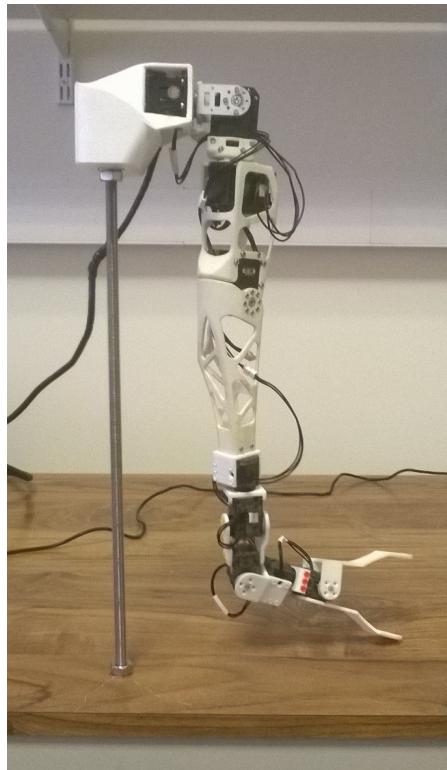


Autres créatures Poppy intéressantes

Un aspect clé du projet Poppy est de stimuler la créativité et l'expérimentation autour de la robotique. Nous essayons de fournir tous les outils nécessaires pour concevoir de nouveaux robots basées sur les mêmes briques technologiques. Quelques nouvelles créatures sont en développement au sein de la communauté. Certaines d'entre elles sont illustrées ci-dessous.

Poppy bras droit (travail en cours)

Poppy bras droit est une créature Poppy basé sur le bras droit du robot Poppy Humanoid, avec 3 moteurs XL-320 supplémentaires pour améliorer la portée et l'agilité du bras. Il utilise le même outil de préhension utilisé par l'Ergo Jr, conçu pour attraper des objets simples.



Le projet a été réalisé lors d'un stage au sein de l'équipe Flowers de l'Inria par [Joel Ortiz Sosa](#). Pour plus d'informations et les sources jetez un coup d'oeil au [répertoire](#).

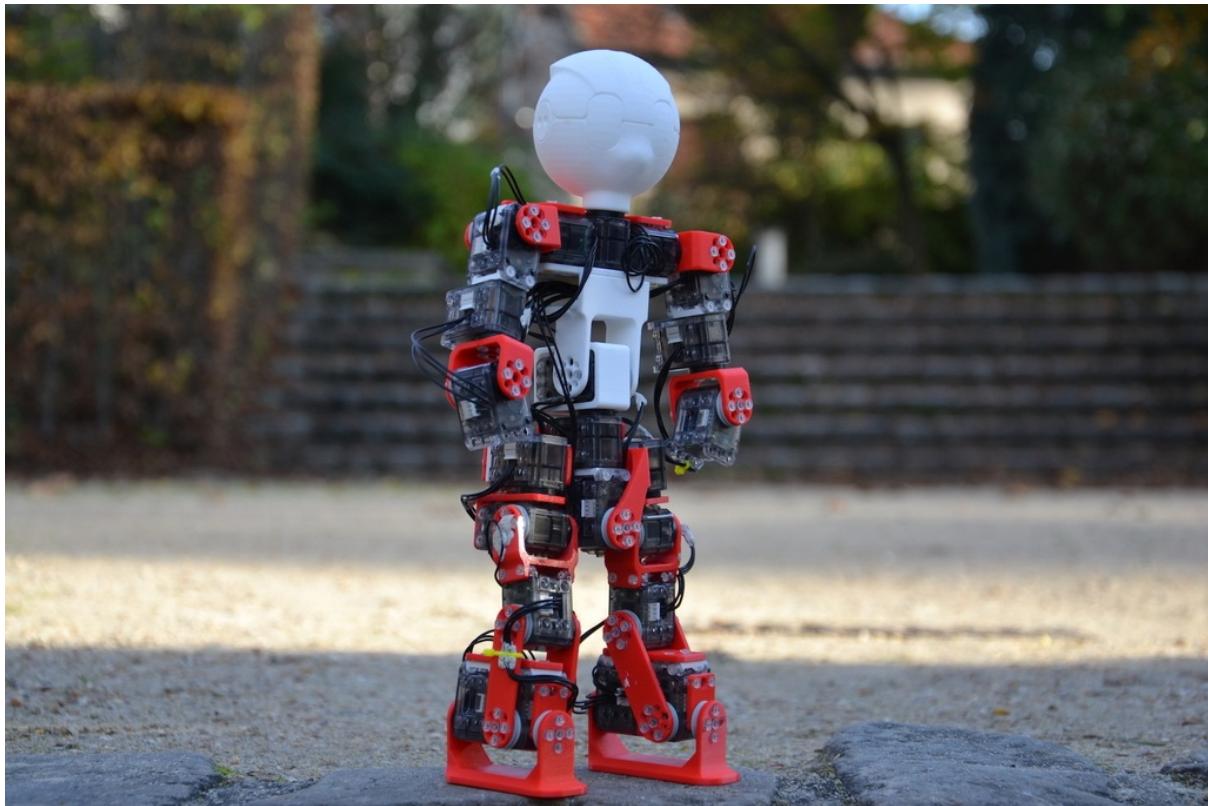
Des Humanoid plus petits et moins chers

Heol

Heol - « soleil » en Breton - est un robot humanoïde haut de 34cm fait par l'association [Heol robotique](#). Composé de 23 moteurs, toutes les autres pièces sont imprimés en 3D. Il utilise également la bibliothèque pypot pour ses mouvements.

Le but de Heol est de donner le sourire aux enfants malades. Il peut être un outil pédagogique en devenant une plateforme d'apprentissage pour la conception mécanique et de programmation.

Sa participation à la RoboCup (coupe du monde de football pour robots) est également envisagée.



Poppyrate

C'est un robot basé sur le Poppy Humanoid. Il vise à développer une version plus abordable grâce à sa petite taille et l'utilisation de moteurs encore moins chers. La réduction de la taille rend également plus facile l'impression des pièces avec une imprimante 3D standard. L'objectif donné est de rendre le robot aussi mobile et customisable que possible tout en maintenant sa compatibilité avec la plateforme Poppy.

Poppyrate sera vendu en kit (avec ou sans les pièces 3D). Il a été conçu par l'entreprise ZeCloud.



Pour plus d'informations, consultez leur [site Web](#) - [Twitter](#) - [Facebook](#) !

Installation

En fonction de ce que vous voulez faire, toutes les sections ci-dessous supposent que vous avez [le client Zeroconf / Bonjour](#) d'installé sur votre ordinateur. Ce n'est pas obligatoire mais autrement, vous devrez connaitre l'adresse IP de votre robot.

Le projet Poppy est vaste, il y a beaucoup de possibilités de chemins d'installation en fonction de ce que vous voulez faire.

Vous voulez installer un robot Poppy physique

Les robots Poppy sont contrôlées par un petit ordinateur embarqué : un Raspberry Pi ou une carte ODROID. Le système d'exploitation de ces ordinateurs est stocké sur une carte SD (vous pouvez également utiliser un contrôleur MMC pour la carte ODROID).

Vous avez deux possibilités : - **vous avez déjà une carte SD préchargée avec le système d'exploitation Poppy (fourni par Generation Robots). Vous n'avez rien à installer, vous êtes prêt à aller vers la section assemblage.** C'est le cas le plus fréquent. - Vous avez une carte SD vierge, vous devez donc [télécharger](#) et [graver](#) le système d'exploitation sur la carte SD.

Si vous êtes familier à Linux et que vous voulez essayer vous-même nos scripts d'installation (qui peuvent être instable), vous pouvez vous rendre au chapitre [installation d'une carte Poppy](#).

Vous voulez essayer des robots Poppy dans un simulateur ou dans le visualiseur web

- Installation des programmes Poppy sur votre ordinateur
- Installation du simulateur V-REP

Vous voulez réaliser des travaux avancés avec un robot tangible

Si vous voulez installer vous-même le système d'exploitation des robots Poppy avec nos scripts d'installation instables : - [Installation d'une carte Poppy](#)

Si vous voulez contrôler un robot à partir de votre ordinateur personnel, vous devez: -[installer des logiciels Poppy sur votre ordinateur](#) -[Installer les drivers USB vers série](#) si vous êtes sous Windows

Zeroconf / Bonjour

Zeroconf également appelé Bonjour (nom donné par Apple) est un ensemble de technologies qui facilite la communication entre les ordinateurs sans configuration préalable.

Zeroconf n'est pas obligatoire sur votre ordinateur pour faire fonctionner les robots Poppy, mais nous allons supposer qu'il est installé. C'est plus commode et plus lisible pour la documentation.

Installation

Windows

Vous devez installer les [Services d'impression Bonjour pour Windows](#) (Oui, c'est un logiciel Apple).

Si vous avez déjà installé un logiciel de la marque Apple comme iTunes ou QuickTime, Bonjour doit déjà être installé.

Parfois, même si Bonjour est déjà installé sur votre ordinateur, vous ne pouvez pas vous connecter directement à votre Ergo Jr. Pour résoudre le problème, désinstallez et réinstallez Bonjour.

Sur les systèmes GNU/Linux

Sur GNU/Linux, vous devez installer *avahi-daemon* (mDNS) et *avahi-autoipd* (IPv4LL), il peut ou ne peut pas être installé par défaut en fonction de votre installation.

Sous Ubuntu/Debian, exécutez

```
sudo apt-get install avahi-daemon avahi-autoipd
```

Sous Fedora / CentOS, exécutez

```
sudo yum install avahi-daemon avahi-autoipd
```

Sous MAC OSX

Bonjour est déjà installé avec OSX. De plus, si vous prévoyez de connecter votre ordinateur directement au robot (sans routeur), utilisez un adaptateur Thunderbolt vers Ethernet plutôt qu'un adaptateur USB vers Ethernet

Vous êtes prêt à continuer votre [parcours d'installation](#).

Ce que Zeroconf fait pour vous

Nom de domaine local (mDNS)

Le client Zeroconf *publie* un nom de domaine local décentralisé (mDNS) avec l'extension de domaine '.local'. Cela signifie que vous pouvez joindre n'importe quel ordinateur local par son nom d'hôte suivit du suffixe '.local' au lieu de son adresse IP.

Avec un client Zeroconf, pour `ping` un ordinateur appelé (hostname) 'ergojr', vous pouvez simplement faire:

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

[...]

Vous n'avez plus besoin de rechercher une adresse IP sur votre réseau local; vous n'avez même pas besoin de comprendre ce qu'est une adresse IP.

Cela fonctionne également sur votre navigateur web. Pour ouvrir le site hébergé sur l'ordinateur du robot appelé 'ergojr', vous devez ouvrir : <http://ergojr.local> dans le champ URL de navigateur web préféré.

Link-local IPv4 addresses (IPv4LL)

Parmi les autres outils de Zeroconf, il existe une implémentation de DHCP décentralisé ([IPv4LL](#)), qui permettent aux ordinateurs d'obtenir une adresse IP et de se connecter les uns aux autres **sans** aucun serveur DHCP.

La gamme [APIPA](#) d'adresse IP automatique se trouve comprise entre 169.254.0.0 et 169.254.255.255.

Vous pouvez brancher un robot à votre ordinateur **directement** sur votre ordinateur avec un câble Ethernet, **sans** aucun routeur et le connecter avec son nom de domaine local (hostname.local).

Vous serez en mesure d'utiliser l'adresse IPv4 de liaison locale **seulement** si vous avez installé vos robots après la fin du mois de mai 2016. Précédemment le paquet *avahi-autoipd* était manquant.

Alternatives pour trouver l'adresse IP d'un ordinateur sur votre réseau local

Si vous ne pouvez pas (ou ne voulez pas) installer le client Zeroconf sur votre ordinateur, vous pouvez utiliser l'une des méthodes suivantes pour trouver l'adresse IP de votre robot.

- Vous pouvez utiliser [Fing](#), célèbre pour ses applications [Android](#) et [iOS](#),
- [Nmap](#) or arp(only GNU/Linux and MAC OSX) if you are not afraid of command line interfaces.

```
nmap -sn 192.168.1.0/24  
arp -an | grep -i B8:27:EB
```

- Vous pouvez également accéder à l'interface Web de votre routeur (avec l'adresse IP sur votre navigateur Web comme <http://192.168.0.1> ou <http://192.168.1.1> ou <http://192.168.0.254> ou encore <http://192.168.1.254>), vous devriez voir une section d'hôtes connectés.

Démarrer avec un robot Poppy

Ce chapitre est réservé aux personnes qui veulent contrôler un robot tangible. Si vous avez l'intention de contrôler un robot simulé par ordinateur, veuillez vous référer au [parcours d'installation pour le simulateur](#).

Les créatures Poppy sont contrôlées par un petit ordinateur embarqué : un Raspberry Pi ou un Odroid.

Le système d'exploitation de cet ordinateur est stocké sur une carte SD (vous pouvez également utiliser une carte MMC pour le Odroid).

Vous pouvez être dans deux types de cas :

- Vous avez déjà une carte SD avec le système d'exploitation Poppy (fournie par l'un des distributeurs de Poppy par exemple).
Vous êtes prêt à passer à la [section montage](#).
- Vous avez une carte SD vierge, il faut donc [télécharger](#) et [écrire](#) le système d'exploitation sur la carte SD.

Le système d'exploitation de créatures Poppy utilise une distribution GNU/Linux, mais vous n'aurez pas besoin de connaissances avancées sur Linux pour installer l'image sur le Raspberry Pi. Vous devez seulement avoir un ordinateur avec un lecteur de cartes SD pour écrire l'image sur la carte SD.

Télécharger l'image du système d'exploitation

Il faut choisir l'image (fichier en `*.img.zip`) à télécharger selon la créature Poppy et la carte contrôleur souhaitée :

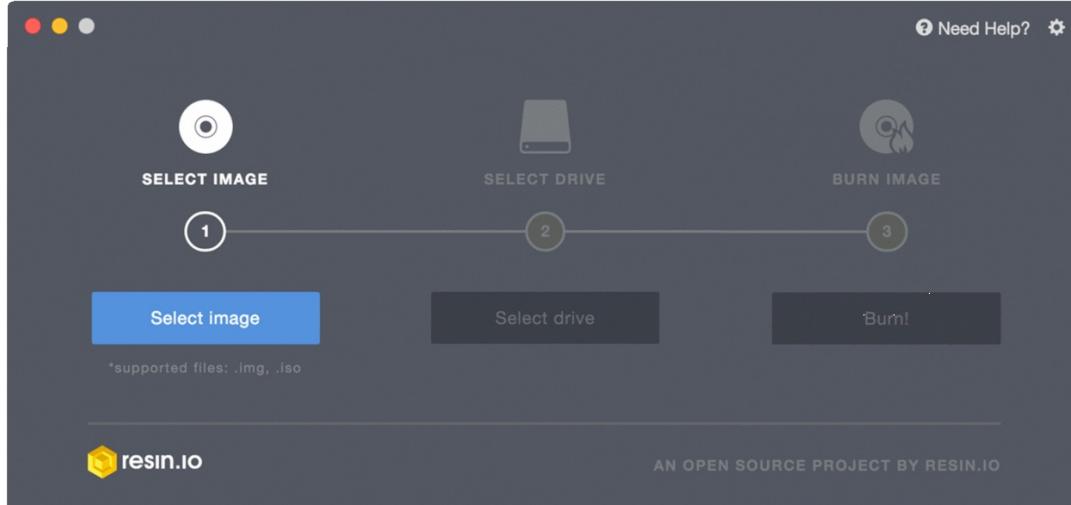
- [Ergo Jr](#)
- [Poppy Torso](#)
- [Poppy Humanoid](#)

Graver l'image de système d'exploitation sur la carte SD

Muni du fichier image correspondant à votre créature Poppy, vous devez utiliser un outil d'écriture pour l'installer sur votre carte SD.

Graver l'image avec Etcher (logiciel avec une interface graphique)

- Téléchargez et installez [Etcher](#). Ce logiciel fonctionne sur les systèmes d'exploitation Windows (versions Windows 7 ou ultérieure), MAC OSX et GNU/Linux.



- Insérez la carte SD dans votre ordinateur. -Démarrez Etcher, sélectionnez le lecteur de carte SD, sélectionnez l'image (un fichier nommé par exemple `2017-04-13-poppy-ergo-jr.img.zip`). Démarrez l'écriture. L'image écrite sur votre carte sera vérifiée automatiquement à la fin de l'opération.

Vous êtes maintenant prêt à [assembler votre robot](#) !

Graver l'image avec `dd` (logiciel en ligne de commande)

Cette méthode fonctionne uniquement pour GNU/Linux et les systèmes d'exploitation OSX et n'est pas recommandée si vous ne comprenez pas ce que vous faites.

Insérez la carte SD et chercher où votre carte SD est montée (`/dev/mmcblk0` et `/dev/disk2` dans l'exemple qui suit). Adaptez à votre cas et exécutez une de ces commandes. Soyez prudent, vous pourriez effacer l'une de vos partitions disques si vous ne comprenez pas ce que vous faites.

Si vous êtes sur un système d'exploitation GNU/Linux :

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

Si vous utilisez OSX ou un autre système d'exploitation basé sur BSD :

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

La commande `dd` ne donne aucune information sur sa progression et donc peut donner l'impression d'être gelée ; cela peut prendre plus de cinq minutes pour terminer l'écriture de la carte. Pour voir l'état d'avancement de l'opération d'écriture, vous pouvez exécuter `sudo pkill -USR1 -n -x dd` dans un autre terminal.

- Exécuter la commande `sync` ; cela garantira que le cache d'écriture est vidé et qu'il est possible de démonter votre carte SD sans causer de corruption des données.
- Retirez la carte SD du lecteur de carte.

Vous êtes maintenant prêt à [assembler votre robot](#) !

Installer le logiciel Poppy

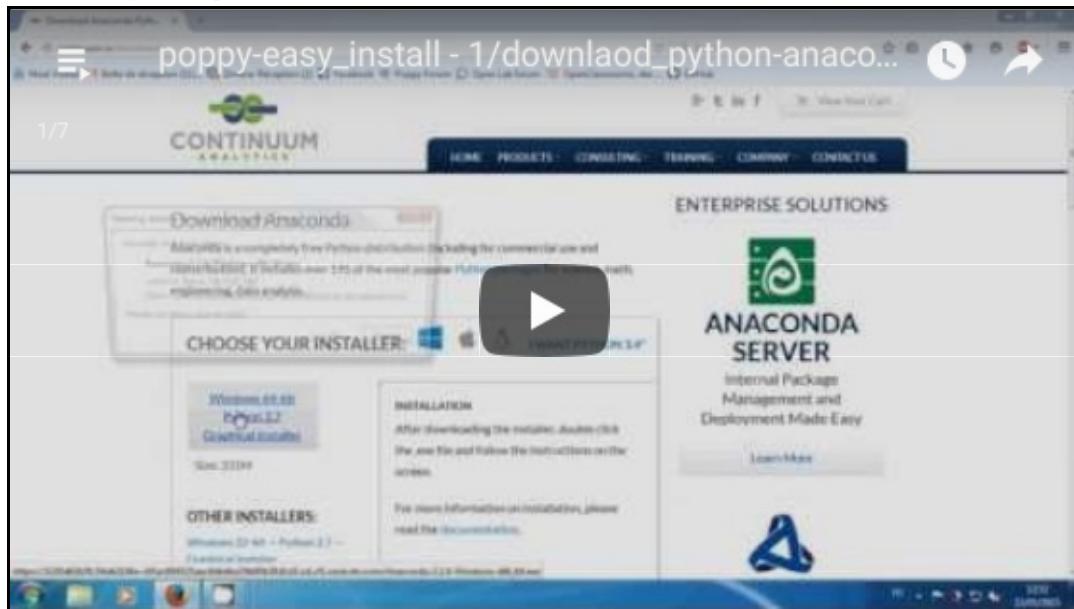
Si vous souhaitez installer le logiciel d'un robot tangible sur une carte embarquée de type Raspberry Pi, allez plutôt au [chapitre de démarrage](#).

Cette section vous guidera pour installer le logiciel Poppy sur votre ordinateur personnel. Elle est utile **seulement** si vous êtes dans l'une de ces situations : *vous souhaitez contrôler un robot simulé*. Vous souhaitez contrôler une créature Poppy depuis votre ordinateur **sans** utiliser la carte embarquée fournie (Odroid ou Raspberry Pi).

Les créatures Poppy sont contrôlées par du code écrit en langage Python. Selon votre système d'exploitation, vous devrez installer Python et dans tous les cas, vous devrez installer les bibliothèques logicielles requises. Si vous faites vos premiers pas avec Python et que vous souhaitez installer un environnement Python complet conçu pour l'informatique scientifique, **nous vous suggérons d'utiliser la distribution Python Anaconda**.

Installer Python et le logiciel Poppy sur Windows

Si vous voulez suivre une vidéo explicative étape par étape de l'installation de Anaconda sur Windows, vous pouvez consulter [ces vidéos](#) (il s'agit d'une playlist YouTube).



Installer Python et les logiciels Poppy sur Windows

Nous encourageons l'utilisation de la distribution Python Anaconda. Toutefois, si vous avez déjà installé une distribution Python telle que Canopy (livrée avec paquets scientifiques), vous pouvez directement [installer les logiciels Poppy](#).

Les bibliothèques logicielles Poppy fonctionnent avec Python 2.7 et Python 3.3+. Si vous n'êtes pas sûr·e de la version à installer, nous vous suggérons d'utiliser Python 2.7, car nous développons avec cette version.

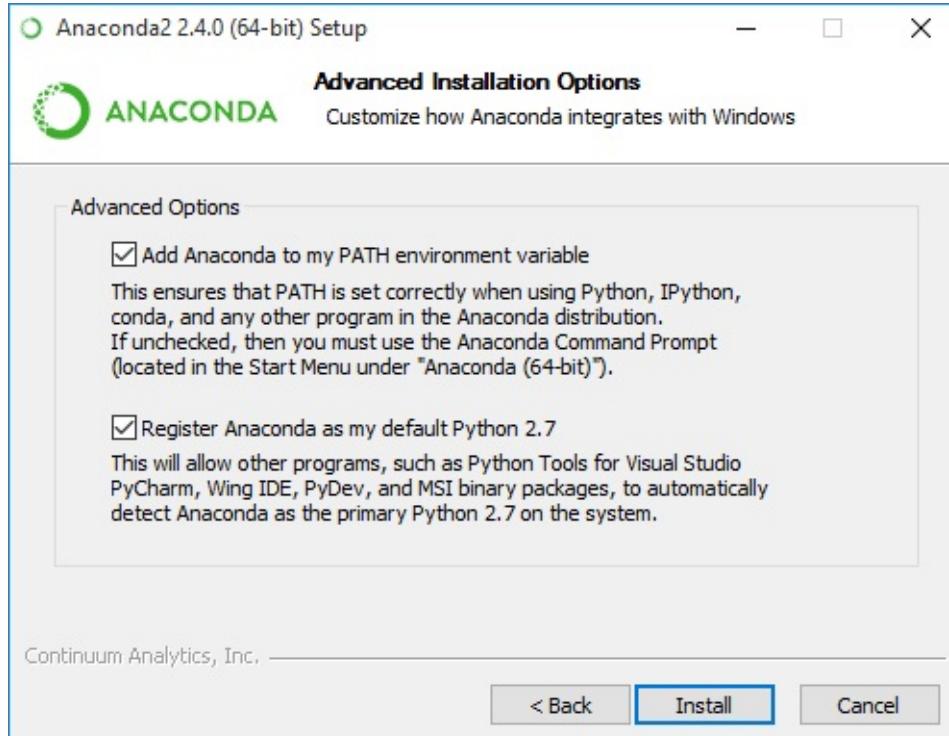
La distribution Anaconda pour Python

Téléchargez la distribution Anaconda pour Python (400 Mo) [ici](#) pour les ordinateurs 64-bit ou [ici](#) pour les ordinateurs 32-bit.

Installez la distribution en cliquant sur « suivant » à chaque étape. Si vous envisagez d'installer Anaconda pour tous les utilisateurs de votre ordinateur, veillez à sélectionner « all users ».



Il est également très important que les deux cases à cocher PATH et "default Python" soient bien cochées.



Maintenant que vous avez une distribution Python, vous êtes prêt à [installer le logiciel Poppy](#).

Python Miniconda (alternative à Anaconda)

Miniconda est une version « allégée » de Anaconda qui contient seulement Python et le gestionnaire de paquets conda. Vous pouvez l'installer **au lieu de Anaconda** et économiser beaucoup d'espace disque (25 Mo vs 400 Mo), mais vous devrez ajouter une nouvelle étape dans le processus d'installation, et vous n'aurez pas de raccourci Jupyter Notebook sur le bureau. Téléchargez Miniconda [ici pour les ordinateurs 64 bits](#) ou [ici pour les ordinateurs 32 bits](#).

Il est également très important que les deux cases à cocher PATH et "default Python" soient bien cochées.

Ouvrez la ligne de commande (appuyez sur les fenêtres principales et tapez « Command Prompt »), tapez et appuyez sur entrée pour exécuter la commande suivante :

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

Vous avez maintenant une distribution Python prête à [installer les logiciels Poppy](#).

Installer les logiciels Poppy sur Windows

Ouvrez la ligne de commande de votre distribution Python (appelée *Anaconda invite* si vous avez installé Anaconda) ou de la ligne de commande de Windows, tapez et appuyez sur entrée pour exécuter la commande suivante :

Remplacez « poppy-ergo-jr » par « poppy-torso » ou « poppy-humanoid » pour installer respectivement un Poppy Torso ou un Poppy Humanoid.

```
pip install poppy-ergo-jr
```

Ceci va installer tout le nécessaire pour contrôler un Poppy Ergo Jr.

Mettre à jour le logiciel Poppy sous Windows

En cas de mise à jour, il est conseillé de mettre à jour pypot (la bibliothèque pour le contrôle des moteurs) et le paquet "creature" séparément :

Remplacez « poppy-ergo-jr » par « poppy-torso » ou « poppy-humanoid » pour installer respectivement un Poppy Torso ou un Poppy Humanoid.

```
<br />pip install pypot --upgrade --no-deps  
pip install poppy-creature --upgrade --no-deps  
pip install poppy-ergo-jr --upgrade --no-deps
```

Pour comprendre les commandes ci-dessus - *--upgrade* désinstallera avant de commencer l'installation - *--no-deps* évitera d'installer les dépendances, c'est utile pour éviter à pip de compiler *scipy* car cela échouera si vous n'avez pas installé les dépendances GCC et Fortran.

Installer Python et les logiciels Poppy sous Mac OSX

Mac OSX est livré avec une distribution Python installée par défaut. Avant d'installer le logiciel Poppy, vous devez installer le gestionnaire de paquets Python **pip**. Ouvrez un terminal, puis appuyez sur entrée pour exécuter la commande suivante :

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

Vous pouvez maintenant installer le logiciel Poppy pour la créature de votre choix :

Remplacez « poppy-ergo-jr » par « poppy-torso » ou « poppy-humanoid » pour installer respectivement un Poppy Torso ou un Poppy Humanoid.

```
pip install poppy-ergo-jr --upgrade
```

Installer Python et les logiciels Poppy sous GNU/Linux

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

Installer la distribution Python Miniconda

Les bibliothèques logicielles Poppy fonctionnent avec Python 2.7 et Python 3.3 +. Si vous n'êtes pas sûr(e) de la version à installer, nous vous suggérons d'utiliser Python 2.7, car nous développons avec cette version.

If you want to have up-to-date numpy, scipy and jupyter without having to compile them, we suggest you to install Anaconda or at least the conda package manager distributed with miniconda. Téléchargez Miniconda (64-bit) avec les commandes ci-dessous dans votre terminal :

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

Si vous avez un ordinateur 32 bits

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

Exécuter les commandes ci-dessous et suivez les instructions pour installer miniconda :

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

```

Vous pouvez maintenant installer quelques dépendances requises ou optionnelles pour le logiciel Poppy avec conda :
```

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

```
Vous pouvez maintenant [installer le logiciel Poppy](#install-poppy-software-on-gnulinux).
```

```
Install dependancies with your operating system package manager (alternative solution to Anaconda/Miniconda)
```

Pypy, 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... La solution pour éviter la compilation de numpy et scipy est de les installer avec le gestionnaire de paquets présent dans votre distribution.

Sur Ubuntu et Debian :

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

Sur Fedora :

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

Sur 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  
sudo pip install jupyter
```

Vous pouvez maintenant [installer les logiciels Poppy](#).

L'inconvénient est que les bibliothèques Python de votre distribution sont très souvent obsolètes.

Installer les logiciels Poppy sous GNU/Linux

Ouvrez un terminal, puis appuyez sur entrée pour exécuter la commande suivante :

Remplacez « poppy-ergo-jr » par « poppy-torso » ou « poppy-humanoid » pour installer respectivement un Poppy Torso ou un Poppy Humanoid.

```
pip install poppy-ergo-jr --user
```

Ceci va installer tout le nécessaire pour contrôler un Poppy Ergo Jr.

Mettre à jour le logiciel Poppy sous GNU/Linux

En cas de mise à jour, il est conseillé de mettre à jour "pypot" (la bibliothèque pour le contrôle des moteurs) et le paquet "creature" séparément :

Remplacez « poppy-ergo-jr » par « poppy-torso » ou « poppy-humanoid » pour installer respectivement un torse Poppy ou un humanoïde Poppy.

```
<br />pip install pypot --upgrade --no-deps  
pip install poppy-creature --upgrade --no-deps  
pip install poppy-ergo-jr --upgrade --no-deps
```

Pour comprendre les commandes ci-dessus - --user va installer le paquet Python dans des répertoires utilisateur, cela évite d'utiliser sudo si vous utilisez l'environnement Python de votre système d'exploitation. --upgrade désinstallera avant de commencer l'installation --no-deps évitera d'installer les dépendances, c'est utile pour éviter à pip de compiler *scipy* car cela échouera si vous n'avez pas installé les dépendances GCC et Fortran

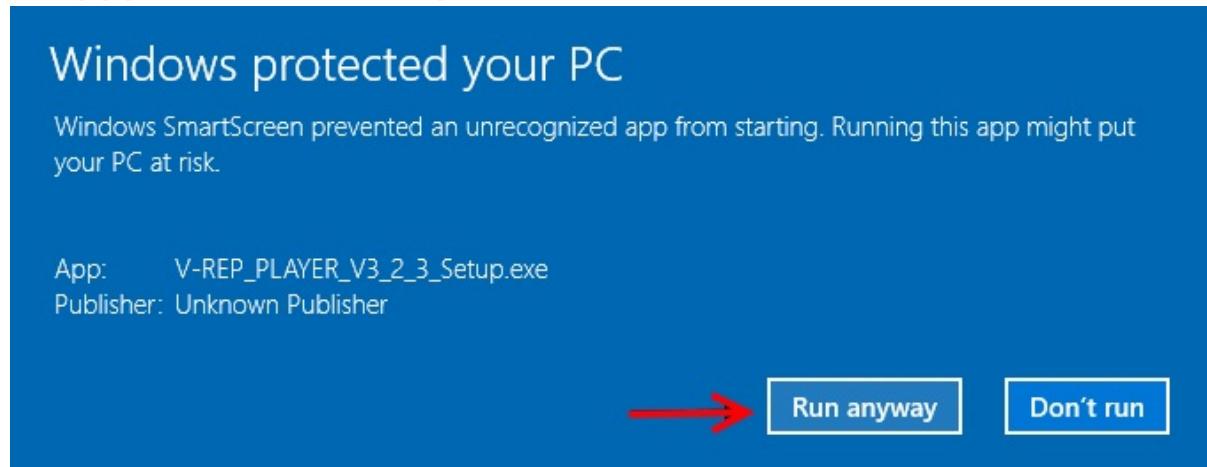
Installation de l'environnement de simulation robotique V-REP

Vous devez installer les logiciels Poppy avant d'installer l'outil de simulation robotique V-REP.

V-REP est un simulateur robotique efficace principalement Open source (GNU GPL), qui est distribué sous une licence gratuite pour les établissements scolaires et propose une licence commerciale pour les autres secteurs. Il existe également une version PRO EVAL qui empêche seulement de sauvegarder. Vous n'avez pas besoin de sauvegarder la scène V-REP pour l'utiliser avec pybot (la bibliothèque Python conçue pour les créatures Poppy), nous vous suggérons donc d'installer cette version sans vous inquiéter d'éventuelles violations de droit d'auteur. Si vous souhaitez modifier la scène V-REP pour ajouter ou personnaliser une créature Poppy, vous devrez utiliser la version PRO ou la version EDU (voir la [licence éducative](#)).

Installation de V-REP sous Windows

Télécharger V-REP PRO EVAL ou EDU (si vous êtes un établissement scolaire). V-REP n'étant pas signé, vous devrez valider la fenêtre popup SmartScreen (sur Windows 10) pour commencer l'installation.



Pendant l'installation, veillez à installer *Visual C++ Redistributable 2010* et *Visual C++ Redistributable 2012*.



Si les versions de *Visual C++ Redistributable 2010* ou *Visual C++ Redistributable 2012* sont déjà présent sur votre ordinateur, il est conseillé de les "réparer" (il s'agit du processus de ré-installation).



Après l'installation, vous pouvez [réaliser un test pour vérifier si V-REP fonctionne bien](#).

Installation sous MAC OSX

Ce paragraphe doit être complété. Votre aide est la bienvenue !

Installation sous GNU/Linux

Ce paragraphe doit être complété. Votre aide est la bienvenue !



Testez votre installation

Ouvrez V-REP avec un double clic sur l'icône du bureau. Ouvrez l'invite de commande de votre distribution Python (appelée *Anaconda prompt* pour Anaconda) ou l'*Invite de commande* de Windows, tapez et validez en appuyant sur entrée pour exécuter la commande ci-dessous :

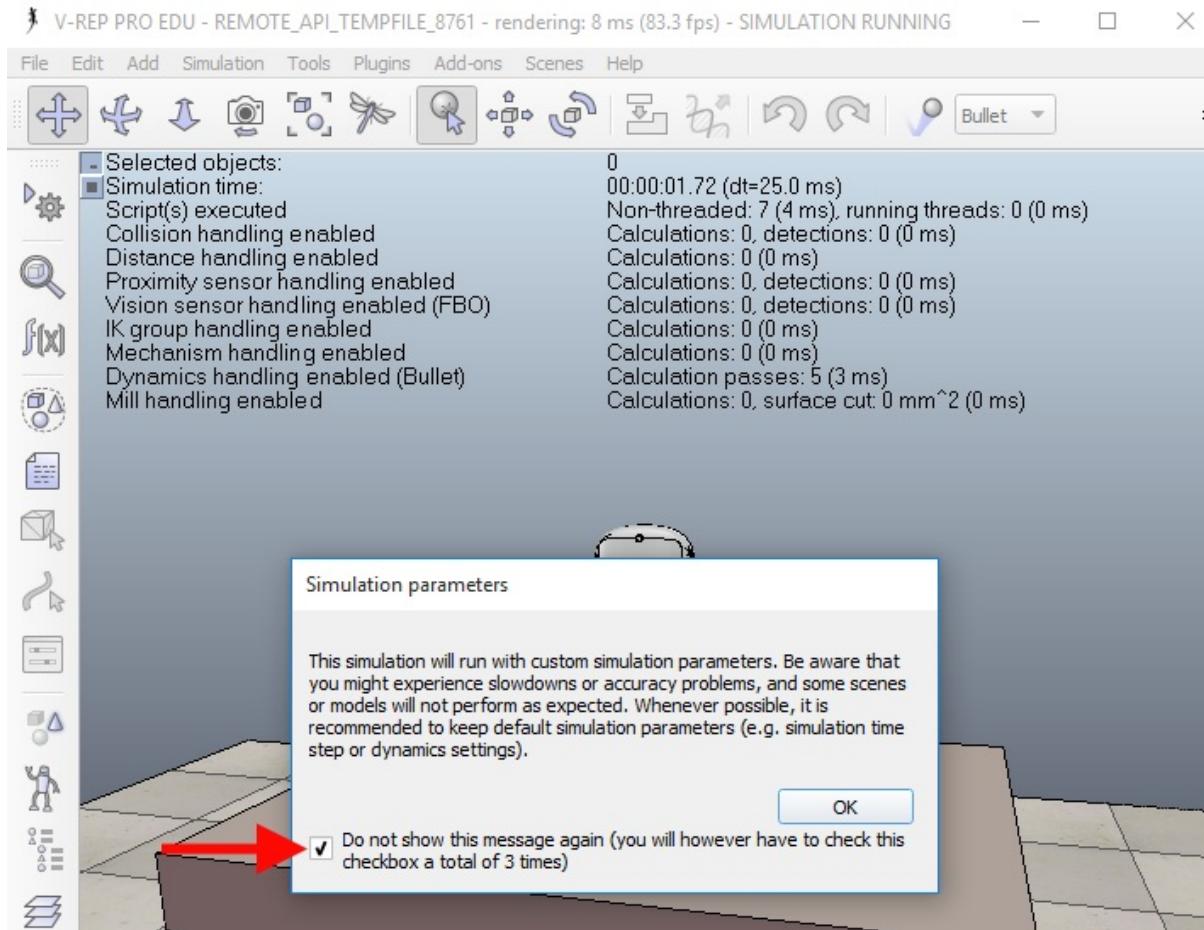
```
poppy-services --snap --vrep --no-browser poppy-torso`
```

Après une ou deux secondes, vous devez avoir une erreur dans votre invite de commande comme le montre l'image ci-dessous.

```
C:\Users\poppy>
C:\Users\poppy>poppy-services --snap --vrep --no-browser
No creature specified, use poppy-torso
Traceback (most recent call last):
  File "c:\users\poppy\miniconda2\lib\runpy.py", line 162, in _run_module_as_main
    "__main__", fname, loader, pkg_name)
  File "c:\users\poppy\miniconda2\lib\runpy.py", line 72, in _run_code
    exec code in run_globals
  File "C:\Users\poppy\Miniconda2\Scripts\poppy-services.exe\_main_.py", line 9, in <module>
  File "c:\users\poppy\miniconda2\lib\site-packages\poppy\creatures\services_launcher.py", line 75, in main
    poppy = installed_poppy_creatures[args.creature](**poppy_args)
  File "c:\users\poppy\miniconda2\lib\site-packages\poppy\creatures\abstractcreature.py", line 86, in __new__
    poppy_creature = from_vrep(config, host, port, scene)
  File "c:\users\poppy\miniconda2\lib\site-packages\pypot\vrep\__init__.py", line 75, in from_vrep
    vrep_io = VrepIO(vrep_host, vrep_port)
  File "c:\users\poppy\miniconda2\lib\site-packages\pypot\vrep\io.py", line 69, in __init__
    self.open_io()
  File "c:\users\poppy\miniconda2\lib\site-packages\pypot\vrep\io.py", line 80, in open_io
    msg.format(self.vrep_host, self.vrep_port))
pypot.vrep.io.VrepConnectionError: Could not connect to V-REP server on localhost:19997. This could also means that you still have a previously opened connection running! (try pypot.vrep.close_all_connections())

C:\Users\poppy>
```

Si vous passez à la fenêtre V-REP, un popup apparaît pour vous informer que la simulation utilise des paramètres personnalisés. Cette popup bloque la communication vers l'API Python de V-Rep. **Vous devez vérifier que la case "Do not show this message again" est bien cochée et appuyez sur "Ok".**



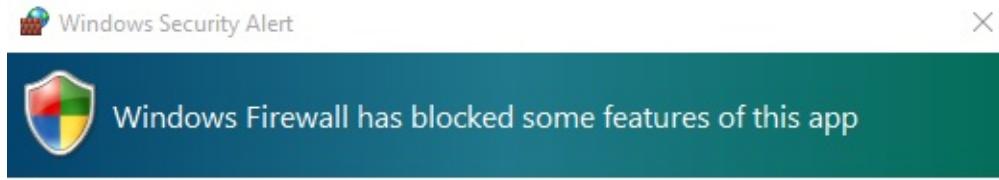
Revenez vers l'invite de commande Windows. Vous devez répéter la dernière commande (`poppy-services --snap --vrep --no-browser poppy-torso`) et cliquez de nouveau sur la fenêtre popup V-REP (avec la case cochée). **Ce processus devra être répété trois fois pour que cela fonctionne bien !**

Pour vous éviter de retaper la même commande encore et encore, vous pouvez appuyer sur la flèche du haut du clavier pour appeler la dernière ligne entrée.

Lorsque la configuration de V-REP est finie, vous pouvez exécuter la dernière commande sans la dernière partie : "--no-browser".

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

Si vous voyez une fenêtre popup de votre pare-feu, comme l'image ci-dessous, assurez-vous de vérifier que "réseau privé" est coché.



Name: python
Publisher: Unknown
Path: C:\users\poppy\miniconda2\python.exe

Allow python to communicate on these networks:

Private networks, such as my home or work network

Public networks, such as those in airports and coffee shops (not recommended because these networks often have little or no security)

[What are the risks of allowing an app through a firewall?](#)

 Allow access

Cancel

Si tout fonctionne bien, un nouvel onglet a été ouvert sur votre navigateur Web par défaut. <-TODO: lien doc-> Vous pouvez débuter la programmation de votre robot dans Snap! ou en Python.



Installer les pilotes

Ce chapitre est destiné aux personnes qui veulent contrôler un robot Poppy **sans** employer une carte embarquée (Raspberry Pi ou Odroid). **C'est un cas particulier pour les utilisateurs avancés.**

Si vous souhaitez contrôler des robots tangibles directement depuis votre ordinateur **sans** un Raspberry Pi ou un Odroid et que vous utilisez un ordinateur avec Windows (vs GNU/Linux ou MAC OSX), vous devrez peut-être installer manuellement les pilotes pour le USB2AX ou le USB2Dynamixel.

Si vous utilisez un **USB2AX**

Si le USB2AX n'est pas reconnu immédiatement (sa LED reste rouge après avoir été branché) sur votre ordinateur, vous devrez probablement installer manuellement ses pilotes. Le processus d'installation et les fichiers à télécharger se trouvent sur la [documentation de USB2AX](#). Vous n'avez pas besoin de pilotes pour GNU/Linux ou MAC OSX, mais notez qu'il ne fonctionne pas très bien avec MAC OSX.

Si vous souhaitez commander des moteurs XL-320 (protocole servomoteurs Dynamixel v2) à partir d'une USB2AX, vous devrez peut-être mettre à jour le firmware à la version 04 de la USB2AX.

Si vous utilisez un **USB2Dynamixel**

Vous devez installer les pilotes FTDI sur votre ordinateur. Vous devez baisser la « valeur de latence du Timer » de 16ms à 1ms (valeur minimum autorisée) comme expliqué dans la [documentation de FTDI](#) pour éviter que les appels pyot fassent un timeout.

Installer un ordinateur embarqué pour un robot Poppy

: ce chapitre est destiné seulement aux personnes qui veulent créer de toutes pièces une image système pour Raspberry Pi ou Odroid. Il est fortement conseillé de simplement [graver une image système fournie par l'équipe Poppy pour installer les logiciels sur votre robot](#).

Pour installer une carte embarquée pour Poppy, nous partons d'une distribution Linux standard (Debian ou Ubuntu), nous supprimons les logiciels inutiles et lançons quelques scripts.

Ayez en tête pas que nos scripts d'installation n'ont pas été conçu pour les utilisateurs finaux : ils ne sont pas forcément à jour et il n'y a presque aucun messages d'erreurs pour vous guider. Si vous rencontrez des problèmes avec ces scripts, vous pouvez poster un message dans la [section d'aide](#) du forum Poppy.

Pour un Poppy Ergo Jr / Raspberry Pi

Les cartes SD Raspberry Pi du commerce sont pré-installées avec le système d'exploitation NOOBS, vous devez tout d'abord installer un système d'exploitation Raspbian.

Téléchargez l'image de votre système : * [Raspbian Jessie](#)

Écrivez l'image sur la carte SD avec un outil d'écriture de disque comme cela est expliqué dans la [section démarrage](#).

Ajoutez un nom de fichier vide « .ssh » dans la partition "boot" pour activer le SSH.

Se connecter sur l'ordinateur embarqué par SSH : `ssh pi@raspberrypi.local`, mot de passe = raspberry.

Remarque : Si vous utilisez Windows, vous n'avez pas de client SSH préinstallé. Vous devez télécharger et installer [putty](#) ou [mobaxterm](#) afin d'utiliser SSH.

Vous devrez vous assurer que vous avez assez d'espace libre sur votre Raspberry Pi. Le moyen le plus simple est d'utiliser le script raspi-config pour étendre votre partition et occuper tout l'espace disque disponible sur la carte SD. Connectez-vous sur votre Raspberry et exécutez (vous devrez redémarrer ensuite) :

```
sudo raspi-config
```

Assurez-vous que votre carte est connectée à Internet et utilisez le programme d'installation « [raspoppy](#) » :

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

Changez `poppy-ergo-jr` dans la commande ci-dessus par la créature Poppy que vous désirez installer sur votre Raspberry Pi.

Redémarrez le Raspberry Pi après la fin de l'installation. Le nom d'hôte, utilisateur par défaut et le mot de passe sera « poppy » (`ssh poppy@poppy.local` mot de passe = poppy). Vous pouvez tester votre installation avec l'interface web dans votre navigateur web en allant sur <http://poppy.local>.

Installer un Poppy Torso / Humanoïde sur un Odroid U3 ou Odroid XU4

Ces ordinateurs embarqués sont équipés de base sur la mémoire MMC d'une image système Ubuntu fonctionnelle, vous pouvez utiliser les scripts d'installation sur celle-ci. Dans le cas où vous n'avez pas de nouvelle installation, vous devez télécharger et graver des images système par défaut suivante: [Ubuntu 14,04 pour Odroid U3](#) [Ubuntu 14,04 pour Odroid XU3/XU4](#)

Pour le graver le système d'exploitation sur la carte MMC/SD, regardez la [section de démarrage](#).

Maintenant que vous avez une installation fraîchement installée, vous pouvez insérer votre carte mémoire dans votre ordinateur embarqué, brancher votre connexion Ethernet, et le mettre sous tension.

Connectez-vous à la carte en SSH: `ssh odroid@odroid.local`, Mot de passe = odroid.

Si vous utilisez Windows, vous n'avez pas de client SSH natif; vous devez télécharger et installer [putty](#) ou [mobaxterm](#) pour utiliser le protocole SSH.

Assurez-vous que votre carte est connectée à Internet, télécharger et exécuter `poppy_setup.sh` (remplacer 'poppy-humanoïde' ci-dessous par 'poppy-torso' si vous voulez installer un robot Poppy Torso):

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

Vous devriez perdre votre connexion SSH en raison du redémarrage de la carte. Ce redémarrage est nécessaire pour procéder à la finalisation du redimensionnement de la partition. Maintenant, votre carte embarqué devrait installer l'environnement Poppy. **S'il vous plaît, n'éteignez ni ne coupez pas l'alimentation de la carte.**

Vous pouvez voir le processus d'installation en vous rebranchant à votre carte avec votre nouveau compte poppy: `ssh poppy@poppy.local` Mot de passe = poppy. **En raison de la compilation de lourds paquets Python (Scipy, Numpy), le processus peut prendre plus d'une heure avant de ce terminer.**

Un processus va automatiquement prendre le contrôle de votre terminal et afficher les informations d'installation. Vous pouvez sortir avec `ctrl + c`. Vous pouvez récupérer les informations en lisant le fichier `install_log`:

```
tail -f install_log
```

Soyez patient ...

À la fin de l'installation, votre ordinateur redémarrera. Vous pouvez consulter le log `tail-f install_log`, si tout s'est bien terminé, les dernières lignes devraient être:

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

Si vous n'êtes pas sûr de ce qui se passe, vous pouvez voir si le processus d'installation est en cours d'exécution avec: `ps up $(pgrep -f 'poppy_launcher.sh')`

Le nom d'hôte, utilisateur par défaut et le mot de passe sera « poppy » (`ssh poppy@poppy.local` mot de passe = poppy). Vous pouvez tester votre installation avec l'interface web dans votre navigateur web en allant sur <http://poppy.local>.

Guide d'assemblage pour l'Ergo Jr



Poppy Ergo Jr est un robot éducatif peu onéreux qui se présente sous la forme d'un petit bras robotique à 6 degrés de liberté. Il se compose de formes très simples qui peuvent être facilement imprimées en 3D. Elles sont assemblées par des rivets qui peuvent être mis ou enlevés très rapidement avec l'outil OLLO.

Son embout peut être facilement changé. Vous pouvez choisir parmi plusieurs outils : *un abat-jour*, une pince, * ou un porte-stylo.

Grâce aux rivets ils peuvent être rapidement et facilement échangés. Cela permet d'adapter l'outil en fonction de vos utilisations du robot.

Les moteurs XL-320 ont les mêmes fonctionnalités que ceux utilisés sur les autres robots Poppy, mais sont légèrement moins puissants et moins précis. L'avantage étant qu'ils sont aussi nettement moins chers.

La carte électronique est visible à côté du robot, ce qui est intéressant pour comprendre, manipuler et brancher des capteurs supplémentaires. Aucune soudure n'est nécessaire, il vous suffit d'ajouter la carte d'extension pixl sur les broches de la Raspberry Pi pour y connecter les moteurs XL-320.

Ce chapitre vous guidera à travers toutes les étapes nécessaires pour assembler entièrement un Poppy Ergo Jr. Il couvrira :

- la configuration des moteurs
- l'assemblage électronique
- l'assemblage des pièces mécaniques

L'assemblage complet ne devrait prendre qu'une heure ou deux la première fois que vous en construisez un. Avec plus de pratique, une demi-heure suffit grandement.

À la fin du tutoriel, vous devriez avoir un Poppy Ergo Jr fonctionnel, prêt à être contrôlé !

Nous vous recommandons de suivre attentivement les instructions. Même si l'Ergo Jr est facilement démontable, il est toujours décevant d'avoir besoin de recommencer une partie de l'assemblage parce qu'on a oublié configurer les moteurs, ou qu'un moteur est inversé.

Liste du matériel

Vous trouverez ici la liste complète du matériel nécessaires pour construire un Poppy Ergo Jr.

Matériel de l'Ergo Jr

- 1 x [carte d'extension Pixl](#) (*carte électronique de contrôle des moteurs XL320 depuis une Raspberry Pi*)
- 1 x [disk_support.stl](#) (coupé à la découpe laser) le plan 2D peut être trouvé [ici](#). *Vous pouvez également fabriquer la base avec une imprimante 3D, mais cela prendra beaucoup de temps*
- les pièces imprimées en 3D [au format STL ici](#)
 - 1 x [base.stl](#)
 - 3 x [horn2horn.stl](#)
 - 3 x [side2side.stl](#)
 - 1 x [long_U.stl](#)
 - 1 x [short_U.stl](#)
 - 1 x [support_camera.stl](#)
 - les différents outils
 - 1 x [lamp.stl](#)
 - 1 x [gripper-fixation.stl](#)
 - 1 x [gripper-fixed_part.stl](#)
 - 1 x [gripper-rotative_part.stl](#)
 - 1 x [pen-holder.stl](#)
 - 1 x [pen-screw.stl](#)

Pièces faites par Robotis

- 6 x servomoteurs dynamixel XL-320
- 1 x jeu de rivets OLLO (vous aurez besoin d'environ 70 rivets colorés et de 4 rivets gris)
- 1 x outil OLLO

Visserie

- 4 x M2.5x6mm vis (pour fixer la Raspberry Pi sur le socle)
- 4 x M2x5mm vis (pour fixer la caméra)
- 4 écrous x M2 (fixation caméra)
- 1 x entretoise M2.5 mâle/femelle 10mm

Divers Electronique

- 1x Raspberry Pi 2 ou 3
- 1x micro SD 8Go (ou plus)
- 1x caméra Raspberry Pi
- 1 x alimentation 7.5V 2A avec un connecteur 2.1 x 5.5 x 9,5 ([celle-ci](#) par exemple).

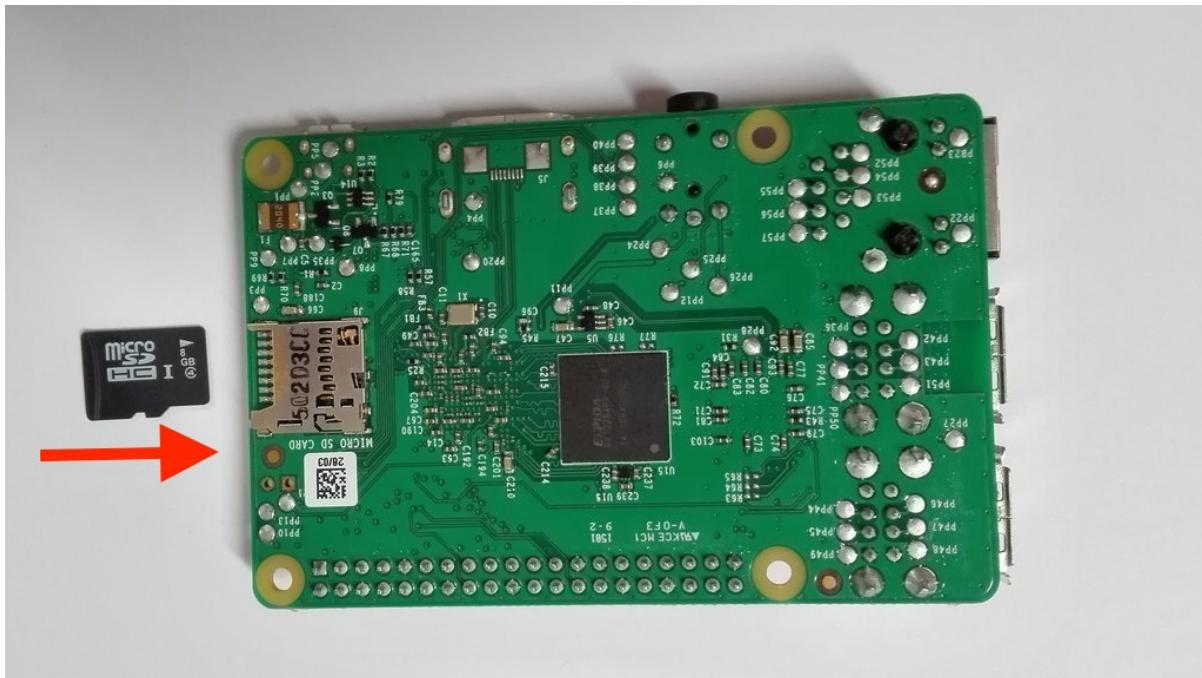
- Câble Ethernet court

Assemblage électronique

Insérez la carte microSD dans la Raspberry Pi

Assurez-vous que vous utilisez une carte micro SD pré-configurée . Si ce n'est pas le cas, vous devez "graver" votre carte micro-SD avec l'image ISO d'Ergo Jr, ce qui est expliqué dans la [section démarrage](#).

Insérez la carte micro-SD à l'intérieur de la Raspberry Pi : poussez la carte micro-SD dans la fente du connecteur jusqu'à entendre un « clic ».

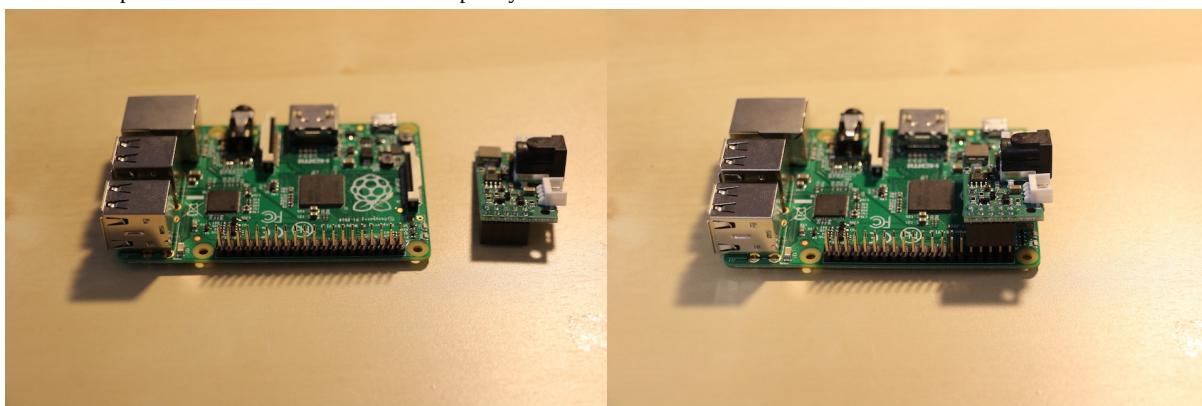


Assemblez la carte d'extension pixl

La carte Pixl peut être achetée sur le site de [Génération Robot](#).

La carte [pixl](#) vous permet d'alimenter la Raspberry Pi à partir d'une alimentation 7.5V DC ou avec des batteries, et vous permet de communiquer avec les moteurs XL-320.

Branchez la pixl à l'extrême des headers du Raspberry Pi.



Une fois que la pixl est branchée (**et pas avant**), vous pouvez brancher l'alimentation et les fils des moteurs.



Vous devez absolument éteindre l'alimentation de la carte d'extension pixl avant de la connecter ou de la déconnecter de la Raspberry pi. Sinon, vous risquez de griller le convertisseur de tension de la carte Pixl.

Vous pouvez à présent [configurer vos moteurs](#).

Configuration des moteurs

L'Ergo Jr est composé de 6 moteurs XL-320 fabriqués par [Robotis](#). Chacun de ces servomoteurs possèdent une carte électronique lui permettant de recevoir différents types de commandes (sur sa position, sa vitesse, son couple...) et de communiquer avec d'autres servos. Vous pouvez chainer ces servomoteurs entre eux et tous les commander depuis un bout de la chaîne.



Cependant, afin d'être connecté et identifié sur le même bus (de donnée), ils doivent avoir un identifiant unique. Sortis de l'usine ils ont tous la même valeur d'identifiant : 1. Dans cette section, nous vous donnerons plus de détails sur comment vous pouvez définir un nouvel identifiant unique à chacun de vos moteurs.

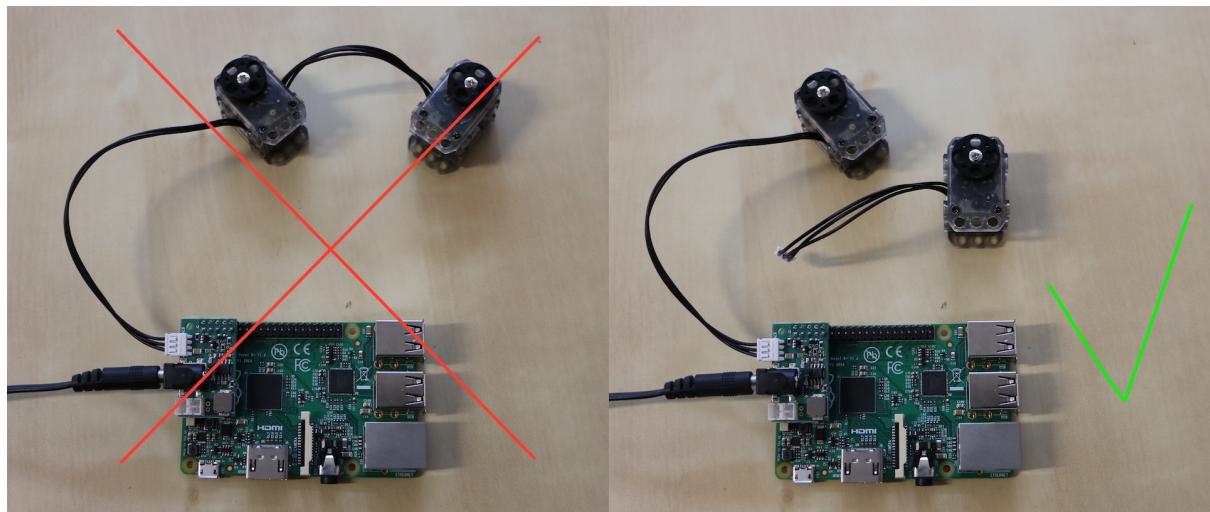
Nous vous recommandons de configurer des moteurs en parallèle de l'assemblage des pièces du robot. Ce qui signifie qu'avant d'assembler un moteur neuf, vous commencez par le configurer puis le monter sur le reste du robot. Pendant la procédure d'assemblage, nous indiquerons chaque fois qu'il faut configurer un nouveau moteur.

Configurer les moteurs un par un

Comme expliqué ci-dessus, tous les moteurs ont le même identifiant par défaut. **Un seul moteur à la fois doit être connecté au bus de données lorsque vous les configurez.** Sinon, tous les moteurs connectés vont penser que l'ordre envoyé sur le bus leur est destiné, ils tenteront d'y répondre ce qui entraînera une "cacophonie" dans les réponses.

Lorsque vous configurez un moteur, vous devriez avoir ces éléments :

- une carte Raspberry Pi
- une carte pixl et son alimentation électrique
- un fil reliant la carte et le moteur que vous souhaitez configurer
- un câble Ethernet reliant la Raspberry Pi et votre ordinateur ou votre routeur

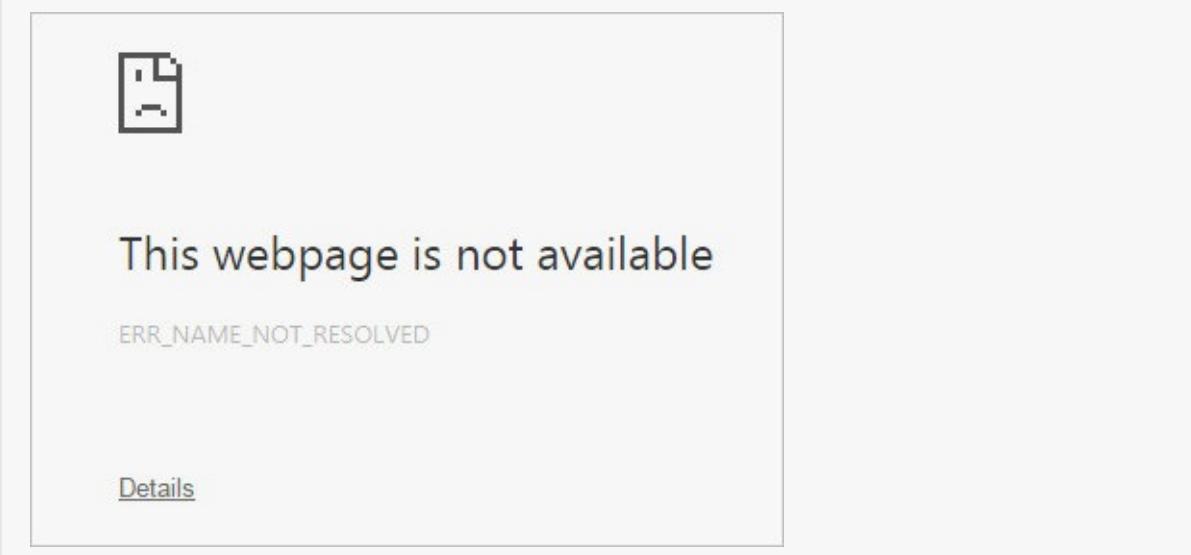


Utilitaire en ligne de commande

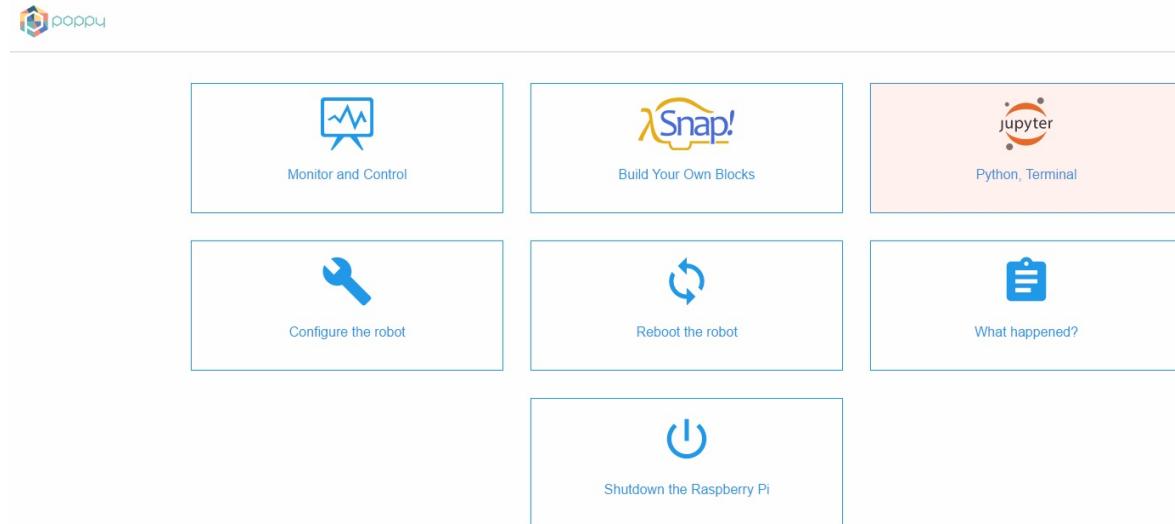
Les robots viennent avec un utilitaire en ligne de commande `poppy-configure`. Pour l'utiliser, vous devrez ouvrir un terminal depuis l'interface Jupyter de votre Raspberry Pi.

Vous pouvez accéder au Raspberry Pi directement à partir de votre ordinateur. Pour ce faire, ouvrez la page <http://poppy.local> dans un navigateur web.

Si vous utilisez Windows, vous devrez installer **Bonjour** (voir [protocole Zeroconf](#) pour plus de détails) afin de pouvoir vous connecter au robot en point à point en utilisant son nom (poppy.local par défaut). Si Bonjour n'est pas installé, vous aurez un message d'erreur comme celui-ci. Si Bonjour est installé et que le problème persiste, veuillez le ré-installer.



Si tout se passe bien, vous devriez voir la page d'accueil du robot :



Cliquez sur le lien "**Jupyter — Python, Terminal**" et après, sélectionnez sur la droite "New" et "Terminal".

The screenshot shows the Jupyter Notebook interface with the following elements:

- Header: Poppy logo and Docs v1.0
- Title bar: Jupyter
- Toolbar: Files, Running, Clusters
- File list: community-notebooks, poppy_src, Benchmark your Poppy robot.ipynb, Discover your Poppy Ergo Jr.ipynb, Record, save and play moves on Poppy Ergo Jr.ipynb, documentation.pdf
- Upload and New dropdown menu: Text File, Folder, Terminal (highlighted), Notebooks, Python 2

Vous avez maintenant accès au Terminal :

The screenshot shows a terminal window with the following content:

```
poppy@poppy:~ $
```

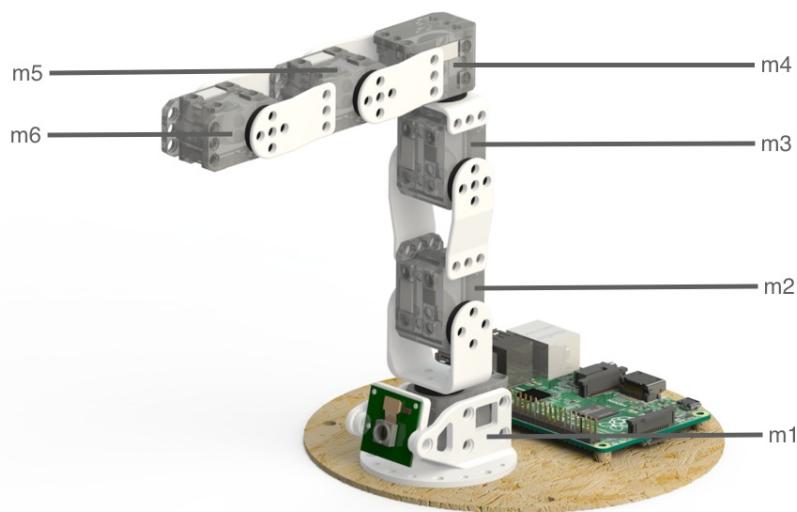
A large pixelated image of a cat is displayed on the screen.

Une fois que le terminal est ouvert, copier et appuyez sur entrée pour exécuter la commande ci-dessous.

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

Vous avez maintenant configuré le moteur m1 de votre robot. Une fois configuré et que vous voyez le message indiquant que tout s'est bien passé, vous pouvez débrancher le moteur (vous n'avez pas besoin de désactiver la carte). La configuration du moteur est sauvegardée dans sa mémoire interne (Eprom).

Les moteurs du Poppy Ergo Jr sont nommés m1, m2, m3, m4, m5, m6. Pour configurer les autres moteurs, il faut changer m1 par le nom du moteur que vous souhaitez configurer dans la commande ci-dessus.



Assemblage mécanique

Avis et avertissements d'ordre général

- Vous pouvez assembler quelques rivets avant la construction. Vous devez insérer la tige de la première partie dans le trou de la seconde. Vous pourrez ainsi les enlever facilement si nécessaire.



Part 1



Part 2



Part 3

- Il y a deux types de rivets : les gris et les autres. Les rivets gris sont plus longs afin de pouvoir les insérer à travers l'axe du moteur, par le côté opposé aux palonniers d'assemblage.



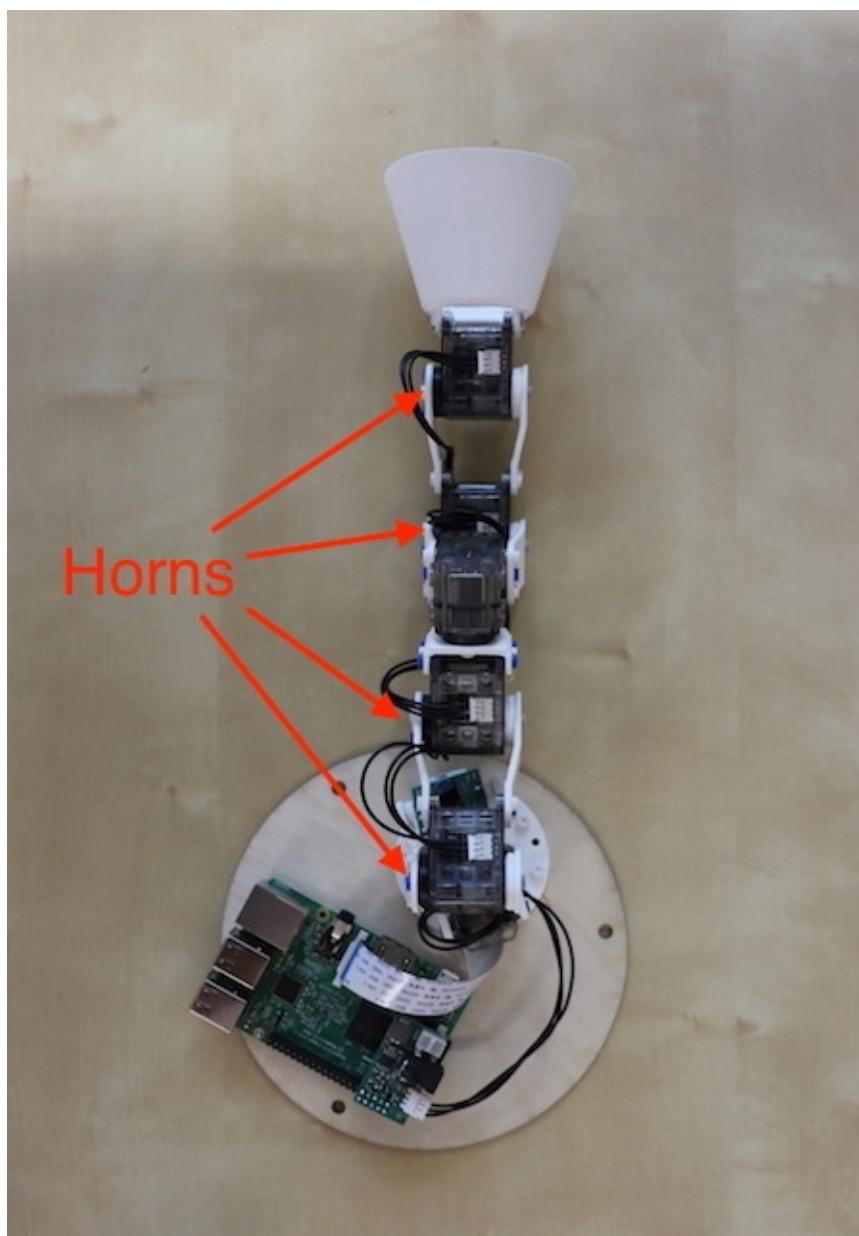
- Utilisez l'outil OLLO pour pouvoir monter et démonter les rivets facilement.



- N'oubliez pas d'insérer les fils entre les moteurs quand vous construisez le robot ! Chaque moteur, mis à part le dernier de la chaîne, doit avoir deux fils : un connecté au moteur précédent et l'autre au moteur suivant (le sens de connexion n'a pas d'importance).
- **Alignez toujours le palonnier (la roue d'entraînement noire) avec le moteur avant de les assembler !** Sinon votre Poppy Ergo Jr aura l'air tout à fait bizarre.

Aligner les palonniers

- Tous les palonniers des moteurs (la roue d'entraînement noire) doivent être alignés **sur le côté gauche du robot**. C'est juste une convention mais elle définira l'orientation de vos moteurs.



Guide pas à pas

Configuration des moteurs (pour toutes les étapes)

Vous pouvez configurer vos moteurs avant, pendant ou après l'assemblage mécanique, mais il est vivement conseillé de configurer chaque moteur un par un dans l'ordre de construction : *configurer le moteur m1* assebmber la base et le moteur m1 *configurer le moteur m2* ...

Pour configurer les moteurs, vous devez les connecter séparément un par un à la Raspberry Pi. Si vous essayez de configurer un nouveau moteur alors qu'il est connecté avec le moteur précédent, cela ne fonctionnera pas.

Pour plus d'informations, consultez la [section configuration de moteur](#).

Étape 1

Tout d'abord, [configurez un moteur XL-320](#) comme « m1 ».

Monter le moteur sur la base imprimée en 3D.



Pour cela, préparez 8 petits rivets. Placez la première partie dans la seconde sans les mettre à l'intérieur du moteur. Ensuite, placez le moteur sur la base avec la palonnier faisant face au côté le plus ouvert. Utilisez l'outil Ollo pour attraper le rivet entre la première et seconde partie puis insérez le rivet dans un des trous d'assemblage. Une fois le rivet en place, verrouillez-le en poussant la première partie du rivet dans la seconde.

Étape 2

Configurez le second moteur, son nom est « m2 », avec la commande suivante dans un terminal du robot :

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

Monter la partie *long_U*. Soyez prudent avec l'orientation de la la pièce, le palonnier doit être orienté vers la gauche. Monter le moteur « m2 » sur le dessus de la construction.



Étape 3

Configurer un troisième moteur : « m3 ».

Monter les pièces *horn2horn* et *horn2side* sur moteur « m2 » et monter « m3 » sur le dessus de la construction.



Étape 4

Configurer le quatrième moteur : « m4 ».

Monter la pièce *short_U* sur le moteur 4.



Monter le moteur « m4 » et la pièce *short_U* préalablement assemblée au sommet de la construction. Le nez du moteur doit être orienté vers l'arrière de la base.



Étape 5

Configurer le cinquième moteur : « m5 ».

Monter les pièces *horn2horn* et *horn2side* sur moteur « m4 » et monter « m5 » sur le dessus de la construction.



Étape 6 - l'outil de votre choix

Configurer le sixième moteur : « m6 ».

Pour terminer votre Ergo Jr, vous devez ajouter un outil à son extrémité. Choisissez un outil en fonction de ce que vous souhaitez faire.

Les outils peuvent être facilement et rapidement changés, ce qui vous permet d'adapter votre robot aux différentes activités.

L'abat-jour ou le stylo

Monter les pièces *horn2horn* et *horn2side* sur moteur « m5 » et monter « m6 » sur le sommet de la construction.



Vous pouvez monter le support de stylo ou de l'abat jour sur le moteur « m6 ».



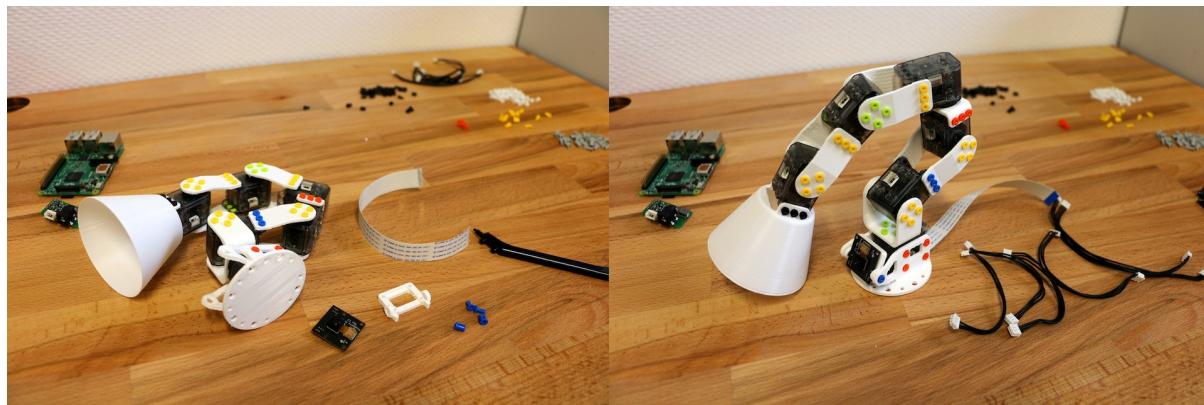
La pince

Monter la pièce *gripper-fixation* entre les moteurs « m5 » et « m6 ».

Monter la pièce *gripper-fixed_part* et *gripper-rotative_part* sur moteur « m6 ».

Étape 7 - électronique

Monter le support caméra sur la base. Fixez la caméra Raspberry Pi dessus et faites passer la nappe de la caméra entre le moteur « m1 » et la base.

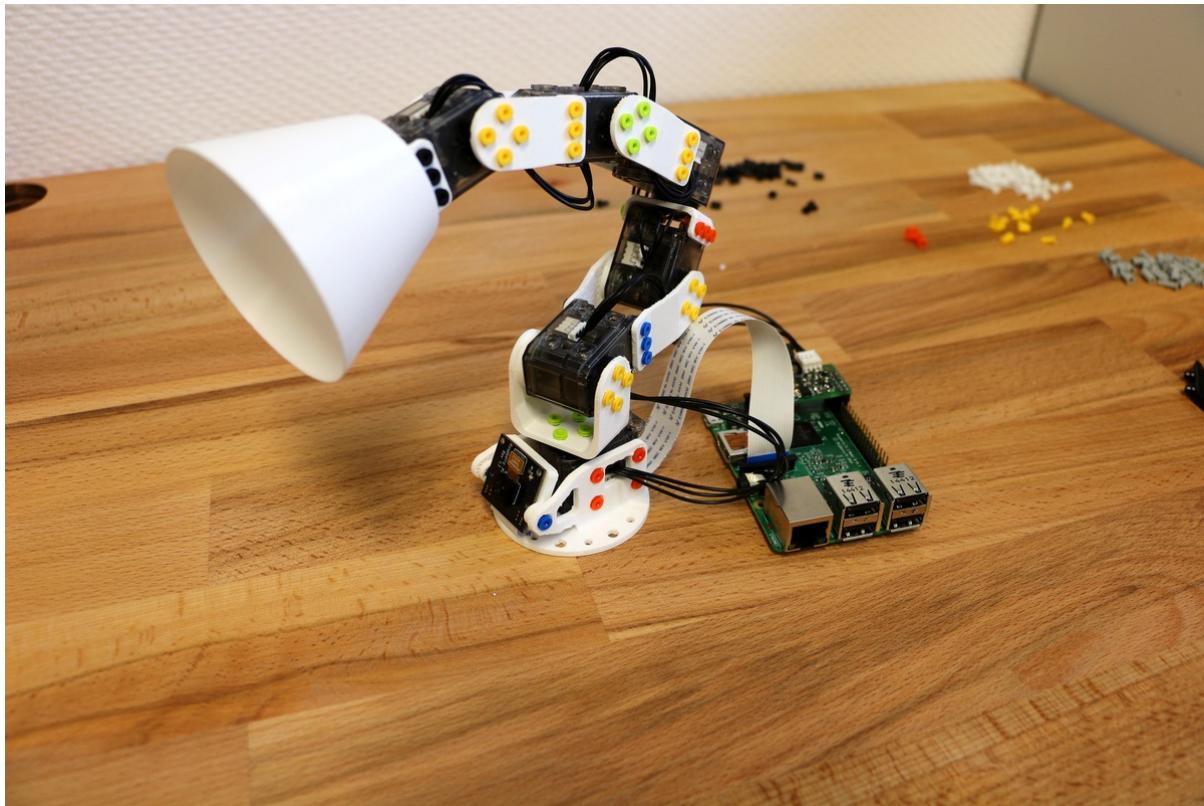


Pour fixer le câble flexible de la caméra sur la Raspberry Pi : *Ouvrez le connecteur caméra en tirant l'attache vers le haut Assurez vous que la partie bleu de la nappe est orientée vers le port Ethernet. * Poussez l'embout de la nappe au fond du connecteur et abaissez la partie supérieure du connecteur pour le fermer*

Les fils des moteurs :

Si ce n'est pas déjà le cas, vous pouvez connecter tous les fils aux moteurs. Chaque moteur a deux connecteurs mais il n'y ni entrée ni sortie, vous devez simplement créer une chaîne de moteurs. Le premier moteur est connecté à la carte d'extension pixel et au second moteur. Le dernier moteur est lié seulement au moteur précédent, et tous les autres sont connectés aux moteurs précédent et suivant.

Les connecteurs du moteur « m1 » (à la base) sont un peu difficiles à brancher, vous pouvez utiliser l'outil OLLO pour vous aider.

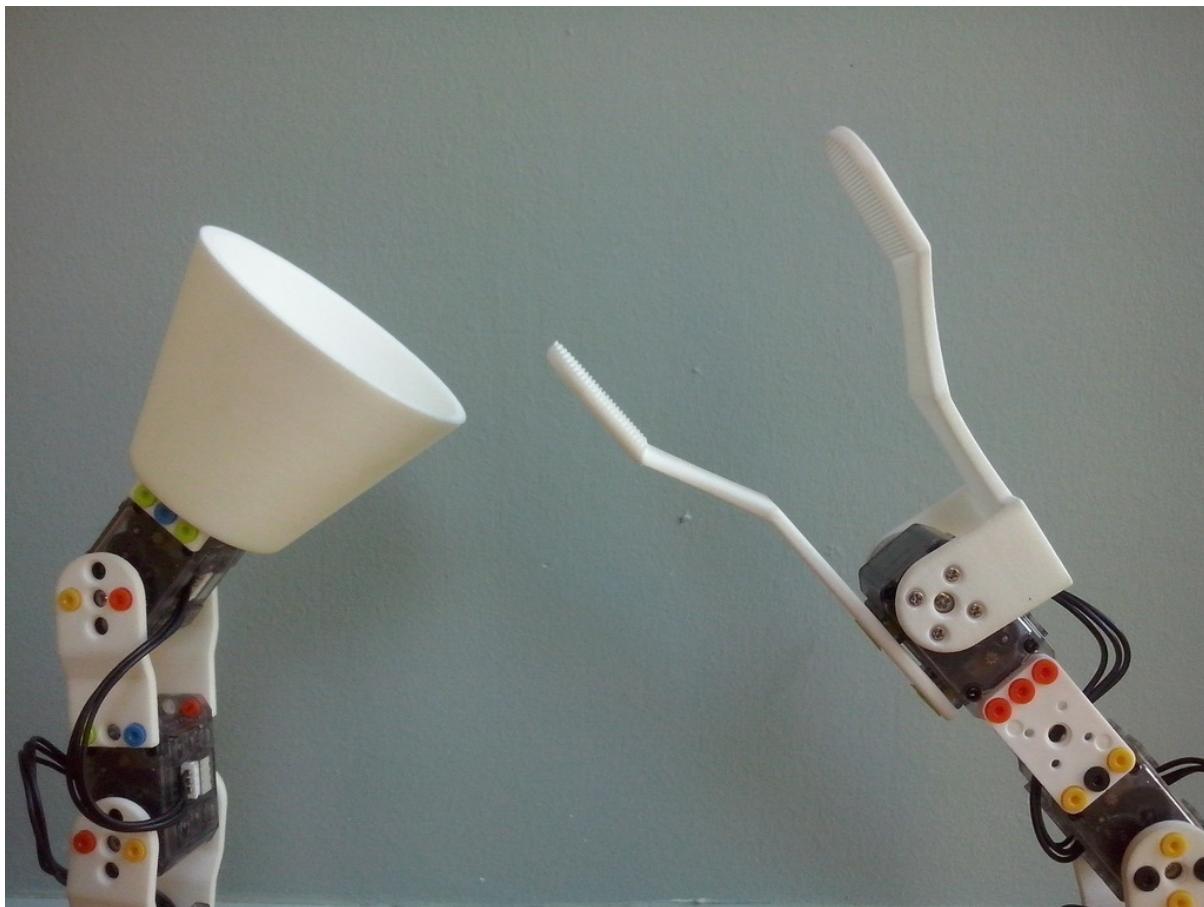


Étape 8 - fixer votre Ergo Jr sur le disque de support en bois

Montez votre Ergo Jr sur le *disque de support* en bois.

Monter votre Raspberry Pi sur le disque de support, et utilisez une vis 4 x M2.5x6mm pour la fixer.

C'est fini !



Vous pouvez désormais vous détendre en buvant [votre boisson préférée](#).

Guide d'assemblage pour Poppy Humanoid

Le guide d'assemblage pour le robot Humanoid n'a toujours pas été fusionné avec nouvelle documentation. Vous le retrouverez sur le [répertoire github poppy-humanoid](#).

Guide d'assemblage pour Poppy Torso

Le guide d'assemblage pour le robot Torso n'a toujours pas été fusionné avec nouvelle doc. Vous le retrouverez sur le répertoire [github poppy-torso](#).

Programming

There is different way to program your Poppy robot. These choices depend on your programming skills and your own 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 you should look at the [Usage of Jupyter notebooks section](#).
- If you want to learn how to use your robot through Python look at the [Programming in Python section](#).
- If you want to remote control your robot with external running 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:

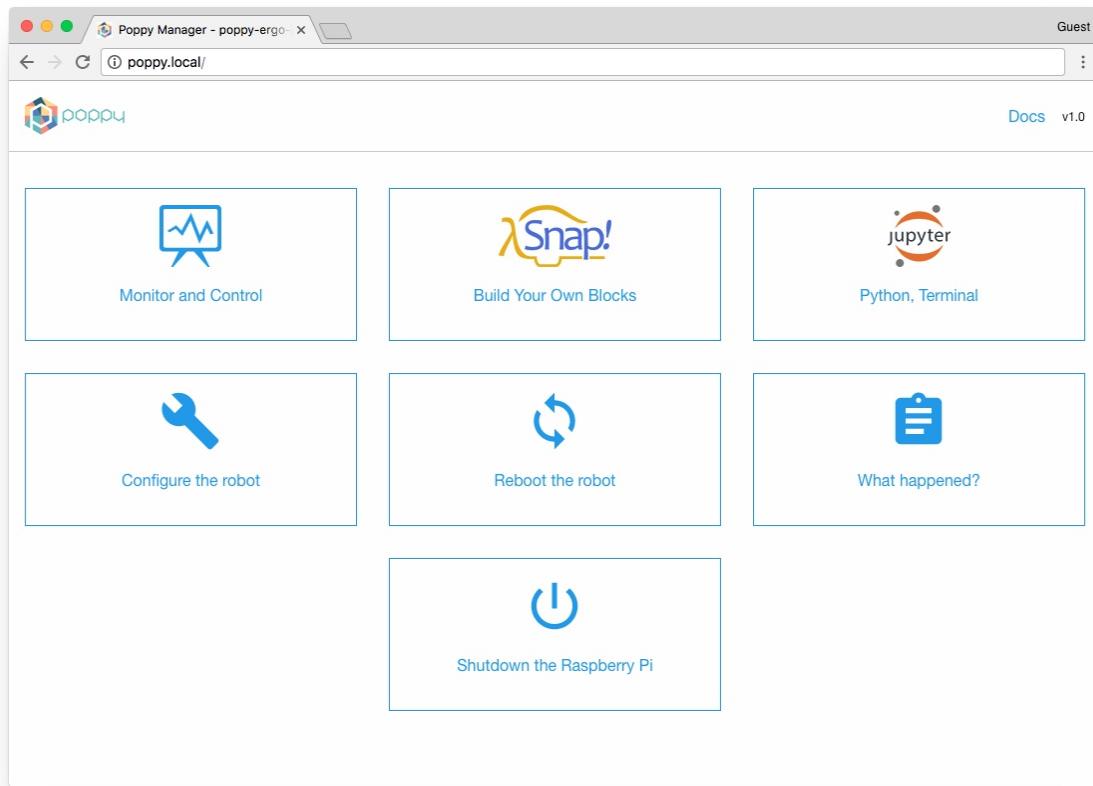
```
from poppy.creatures import PoppyErgoJr
poppy = PoppyErgoJr(simulator='poppy-simu', use_snap=True)
```

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.

 A screenshot of a 'Sign up' modal window. It features a blue header bar with the text 'Sign up'. Below the header is a large blue cloud icon. The form contains fields for 'User name' (with a placeholder 'Type your user name...'), 'Birth date' (with dropdown menus for month and year), and 'E-mail address' (with a placeholder 'Type your e-mail address...'). At the bottom of the form are two buttons: 'Terms of Service...' and 'Privacy...', both with small descriptive text below them. Below these buttons is a checkbox labeled 'I have read and agree to the Terms of Service' with a checked box. At the very bottom are 'OK' and 'Cancel' buttons.

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".

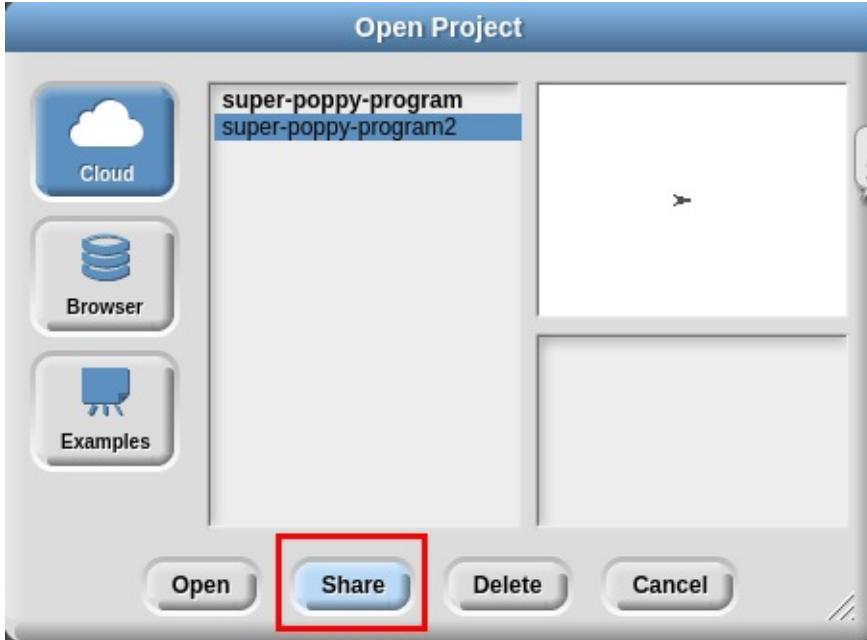


Share your Snap! project

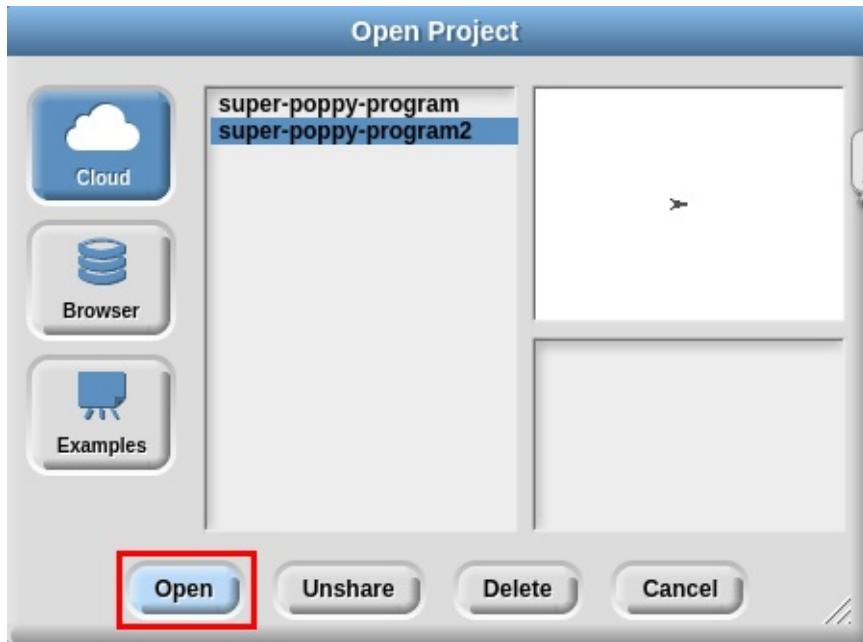
The big advantage of using Snap! Cloud is the ability to share a copy of your project with anyone. To share a Snap! project, you first need to be logged in Snap! Cloud and having your current project saved ("save" or "save as"). Go to the "open" menu:



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.

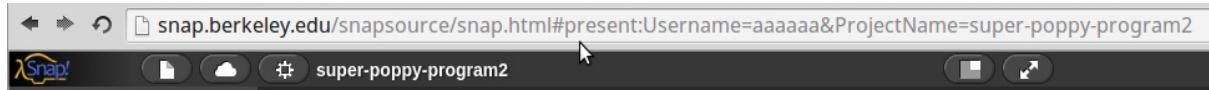


You can copy and paste the URL and share it by the way you want to your friends or to the Poppy community with the forum forum.poppy-project.org.

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.

```

<project name="super-poppy-program2" app="Snap! 4.0, http://snap.berkeley.edu" version="1">
  <notes/>
  <thumbnails>
    data:image/png;base64,iVBORw0KGgoAAAANSUhEUgAAAKAAAB4CAYAAABlovlvAAADCULeQVR4Xu3VMJ5bARRE0c8+TG+WQ8GGLwh1gMLceFIFBENST0eZSR03NrvPfn8K/vhdrvDdI8CI4E
  </thumbnails>
  <stage width="480" height="360" costume="0" tempo="60" threadsafe="false" lines="round" codify="false" inheritance="false" scheduled="false" id="1">
    <pentrails>
      data:image/png;base64,iVBORw0KGgoAAAANSUhEUgAAAAeAAAACPNygAAAOhULEQVR4Xu3VwQkAAAJEMN1/abewn7jAQRCG4wgQIECAAIF3gX1fNEiAAAECAiMAHsCAGQIECAQC
    </pentrails>
    <costumes>
      <list id="2"/>
    </costumes>
    <sounds>
      <list id="3"/>
    </sounds>
    <variables/>
    <blocks/>
    <scripts/>
    <sprites>
      <sprite name="Sprite" idx="1" x="0" y="0" heading="90">
        <costumes>
          <list id="9"/>
        </costumes>
        <sounds>
          <list id="10"/>
        </sounds>
        <variables/>
        <blocks/>
        <scripts>
          <><script x="202" y="295">
            <block s="doGlide">
              <l>1</l>
              <l>0</l>
              <l>0</l>
            </block>
          </script>
          <><script x="127" y="148">
            <block s="forward">
              <l>10</l>
            </block>
            <block s="setHeading">
              <l>90</l>
            </block>
          </script>
        </scripts>
      </sprite>
    </sprites>
  </stage>

```

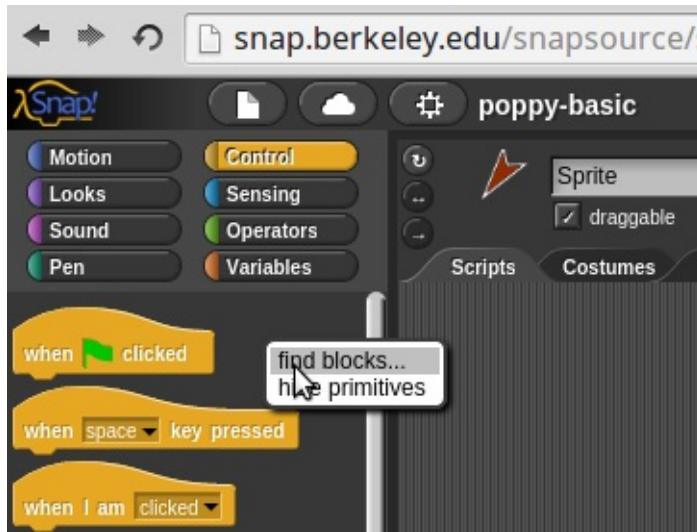
This file describes all your Snap! project in a simple file. It's not made to be human readable so don't be afraid, you just have to save it on your computer. For that, do a right click, choose "save as" and a proper name and location on your computer for this project.

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



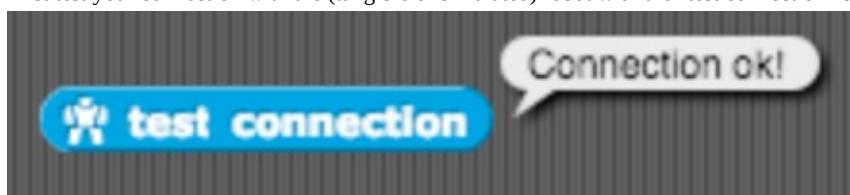
Search Poppy blocks

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.



Réseau

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.

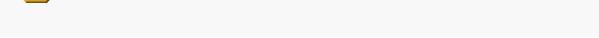
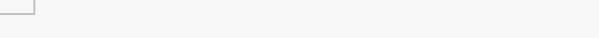
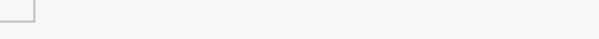


Build your own blocks!

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

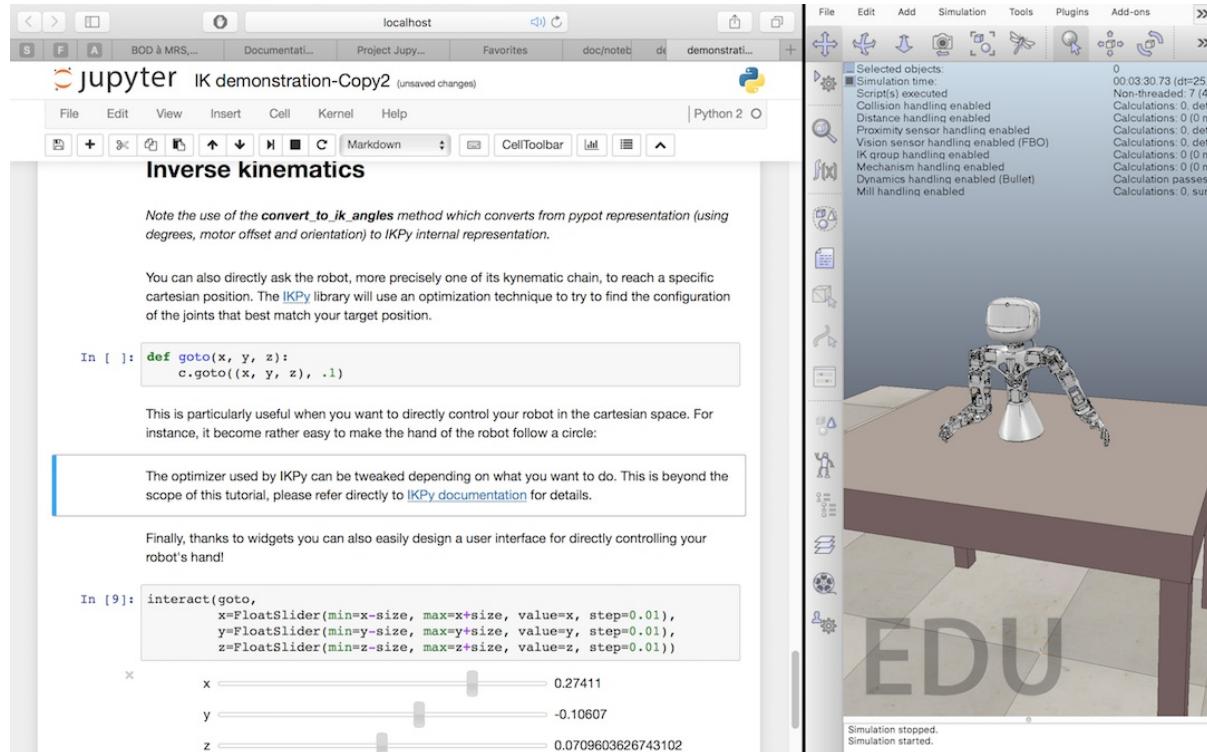
	Description
	This block allows you to connect Snap! to your robot. The host input can accept : - robot_name.local (e.g. : poppy.local if your robot's name is poppy)- the IP address (e.g. 123.124.145.176)
	Click on this block to verify that you are connected to your robot.
	Put one or many motors in compliant or stiff mode. Motors are hand-drivable in compliant mode but must be in stiff mode to be controlled with Snap!. The "motor(s)" input can accept:- string of a motor name (e.g. m1)- string of many motors separated with spaces (e.g. m1 m2 m4)- a Snap! list of motors like the reporter block "\$robot all motors"" or the block "list""
	Put one or many motors in a position (angle in degree) in a given time. The "motor(s)" input can accept: - string of a motor name (e.g. m1) - string of many motors separated with spaces (e.g. m1 m2 m4) - a Snap! list of motors like the reporter block "\$robot all motors"" or the block "list". "Wait" can be true or false. If it's on True, the action will wait the end of the previous action. If it's on False, then the action will proceed during the previous one."
	Restart the software inside the robot.
	Put the value to one register (position, speed, max

	Put the value to one register (position, speed, max torque, ...) of one or many motors.
	Activate/deactivate color leds of motors and choose the color of your choice. The "motor(s)" input can accept: - string of a motor name (e.g. m1) - string of many motors separated with spaces (e.g. m1 m2 m4) - a Snap! list of motors like the reporter block "\$robot all motors"" or the block "list"
	Play a movement at a given speed. It is necessary to indicate the exact name of the movement previously recorded. This block can be nested in the "concurrent/sequential" block."
	All blocks in input are run one after each other. You can use this block to play concurrently many sequentially move.
	All reports input are run simultaneously. You can use this block to play concurrently many recorded move.
	Create and start a movement recorded by demonstration to the given motors.
	Stop a record and save the recorded move in the robot. Be careful, you must have previously defined a move record with the "create & start record move ..." block.
	Stop a movement that is being played.
	Play movements at the same time (can be nested to concurrent block). Inputs can be : - move name (string) - any reporter block, like sequential or play sequentially"
	Play movements following in order (can be nested to sequential block). Inputs can be : - move name (string) - any reporter block, like sequential or play sequentially
	Start/Stop/Pause/Resume an integrated behaviour of the robot. It can be a position, a movement, a sensorimotor loop, high level camera feature..
	Play a move in reverse at a given speed (reporter block)
	Play a move at a given speed (command block)
	Give motors which are in a given group. You can know groups name with the block "all motors groups".
	Return a list with the name of every motors in the robot.
	Give the value of a register (position, speed, load, ...) of one or many motors.
	Give the position of every motors. It is a shortcut to

 get all motors positions	the block above. It is useful to make a snapshot of pose of the robot.
 index of motor <input type="text" value="motor_name"/>	Return the index of a motor name in the "all motors" block list.
 robot URL	Give the URL of the robot. For internal use only.
 all recorded moves	Give all records stored in this robot.
 get all behaviours	Give the list of all attached/running behaviours .
<input type="checkbox"/>	Give all existing motors groups.
 get méthodes of behaviour <input type="text"/>	Get all methods or functions which are runnable in a behavior. It is an advanced block.
 card caribou is detected ?	Return a boolean (true/false) depending if the selected card is detected by the camera of the robot.

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.



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).

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

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.

N'hésitez pas à nous faire savoir s'il manque certaines références d'ordinateurs portables! Vous pouvez directement envoyer une pull-request sur GitHub ou utiliser [le gestionnaire de bug](#).

Programmation des robots Poppy en Python

Ce chapitre vous guidera de manière à pouvoir contrôler les robots Poppy en Python. Comme c'est le langage actuel pour écrire des librairies Poppy, vous verrez comment accéder tous les différents niveaux de contrôle, du plus haut vers le plus bas.

Nous allons détailler tout ce que vous devez savoir pour pouvoir programmer directement le robot en utilisant le Python embarqué dans le robot Poppy ou pour l'installer localement. Remarquez que ce chapitre ne prétend pas vous apprendre le langage Python ou la programmation à partir de zéro et donc si vous êtes totalement nouveau sur le langage Python, il peut être bon de commencer avec un tutoriel de Python. Pourtant, nous essayons de garder les tutoriels aussi simples que possible et nous vous avertirons toujours lorsque certaines parties ciblent les utilisateurs les plus avancés.

Nous allons essayer de fournir autant d'exemples que possible et de l'axer sur l'API complète ainsi vous pouvez trouver et utiliser les fonctionnalités moins courantes. La plupart des exemples et didacticiels sont disponibles comme une liste de [notebooks Jupyter](#). Le prochain chapitre, [Galerie de notebooks Jupyter](#), présente une liste descriptive de chaque notebook, de ce qu'ils contiennent, comment ils peuvent être utilisés, pour quel robot, etc.

Toutes les bibliothèques Poppy sont open source et sont distribués sous la licence [GPL v3](#). Ainsi, vous pouvez accéder librement le code source sur [GitHub](#). N'hésitez pas à créer un fork, envoyer un pull/request et à contribuer !

Pourquoi Python et Anaconda ?



```
print("Hello, world!")
```

Les bibliothèques développées pour le projet Poppy ont été conçus dans le but de rendre facile et rapide l'écriture du code de contrôle des différents robots basés sur - à l'origine - les servomoteurs dynamixel robotis. L'idée était de fournir un accès depuis le bas niveau -communication série brute avec un moteur spécifique par exemple- à des niveaux plus élevés tels que le démarrage et l'arrêt de primitives/comportements (p. ex. suivi de visage, postures,...) ou l'enregistrement directement des mouvements grâce à de l'apprentissage par démonstration.

Nous avons décidé d'écrire la plupart d'entre eux en Python, car sa souplesse permet un développement rapide et modulaire. Elles étaient également destiné à être accessible par un large public, de développeurs, de roboticiens en général, pour des amateurs, des chercheurs, des artistes... Python a également été choisi pour l'énorme choix de bibliothèques existantes (scientifique, vision par ordinateur, web...), donc si quelqu'un souhaite ajouter un nouvel élément, comme le support à un nouveau moteur/capteur, cela devrait être réalisable le plus facilement et rapidement possible.

Enfin, le support multiplateforme et la facilité d'installation étaient également des aspects essentiels.

Nous conseillons fortement d'utiliser la [distribution Python Anaconda](#) puisqu'elle comprend déjà la plupart des bibliothèques nécessaire aux bibliothèques Poppy. Nous fournissons également toutes les bibliothèques Poppy comme conda recipes afin d'être facilement installable à l'aide de Anaconda (voir [section d'installation](#)).

Vue d'ensemble des différentes bibliothèques

Une documentation plus détaillée de ces bibliothèques logicielles est disponible dans la [section bibliothèques logicielles](#)

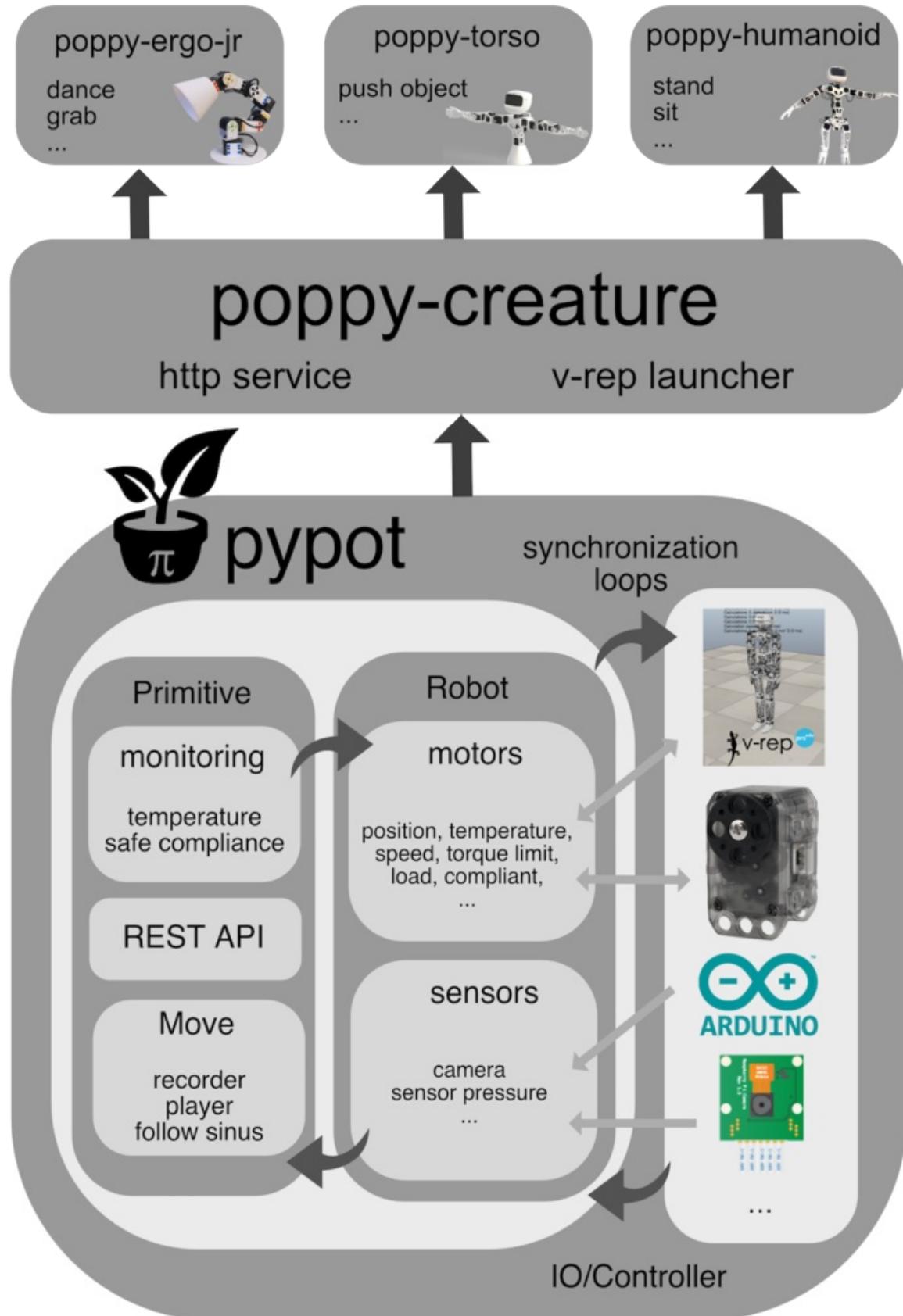


Avant de se lancer dans la programmation, nous allons présenter brièvement les différentes bibliothèques Poppy existantes et comment elles interagissent entre elles.

Il y a trois principales bibliothèques logicielles :

- [pypot](#) : C'est le cœur de l'architecture logicielle Poppy. Pypot gère toutes les communications de bas niveau avec le matériel (capteurs et moteurs), définit les boucles de synchronisation afin que votre commande soient toujours à jour. Il fournit également les primitives du mécanisme qui permet la définition d'un comportement simple qui peut être -plus ou moins- automatiquement combinés.
- [poppy-creature](#) : Cette bibliothèque définit les outils communs partagés par tous les robots Poppy, par exemple comment faire pour lancer le simulateur ou démarrer l'API HTTP automatiquement attaché à n'importe quel robot.
- [poppy-ergo-jr](#), [poppy-torso](#), and [poppy-humanoid](#): ces bibliothèques sont spécifiques à chaque robot Poppy. Elles définissent la configuration particulière du robot, les capteurs utilisés, quels moteurs sont connectés à quel bus... C'est aussi ici que des comportements spécifiques à une créature sont définis (la primitive qui fait Poppy Humanoid se tenir debout par exemple).

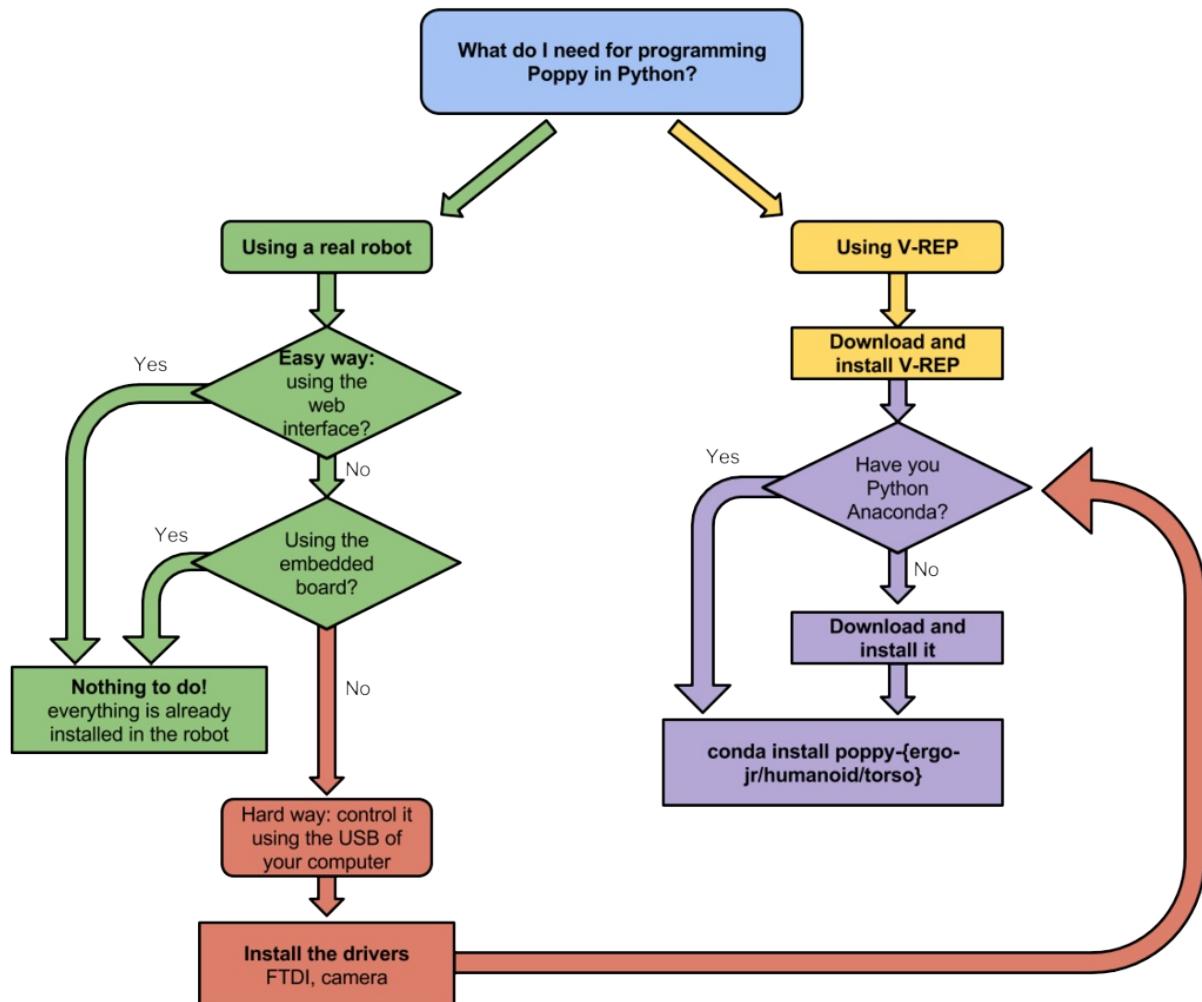
Cela est résumé dans le schéma ci-dessous :



Installation

Tout d'abord, notez que si vous n'envisagez d'utiliser que des robots réels, ils viennent avec Python et toutes les bibliothèques Poppy déjà installées. Vous pouvez vous connecter directement au serveur Jupyter Notebook via l'[interface web](#) et n'avez rien à installer sur votre machine !

Ce que vous devez installer est résumé dans le schéma ci-dessous :



Ainsi, si vous prévoyez de soit

- Utiliser un simulateur (p. ex. V-REP ou simulateur web),
- ou de brancher le robot à votre ordinateur

Vous devrez installer les bibliothèques Poppy localement. Elles fonctionnent sur Windows, Mac OSX, Linux et ont été testées sur :

- 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(simulator='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
```

Vous pouvez ensuite instancier la créature poppy-ergo-jr :

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

Cela va créer un serveur pour Snap ! sur port 6969 et un serveur pour le visualiseur sur le port 8080.

Vous pouvez ensuite vous diriger vers la [page du visualiseur](#).

Accéder aux capteurs et aux moteurs

L'objet robot que vous venez de créer contient deux groupes principaux d'objets :

- moteurs
- sensors

auxquelles on peut facilement accéder à l'aide de `poppy.motors` et `poppy.sensors`. Dès que l'objet robot est créé, il débute automatiquement des boucles de synchronisation qui assureront que les dernières valeurs disponibles sont reçus/envoyés au robot.

Les servomoteurs qui sont utilisés dans des robots Poppy peuvent être considérés à la fois comme des moteurs ou des capteurs. 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 compliant:

```
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 your 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 were 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 are 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

Cette page est encore vierge. Votre aide est nécessaire pour la remplir !

Galerie d'activités

Cette page est encore vierge. Votre aide est nécessaire pour la remplir !

Contrôler Poppy avec un Arduino via Snap4Arduino

Rédigé par [Gilles Lassus](#).

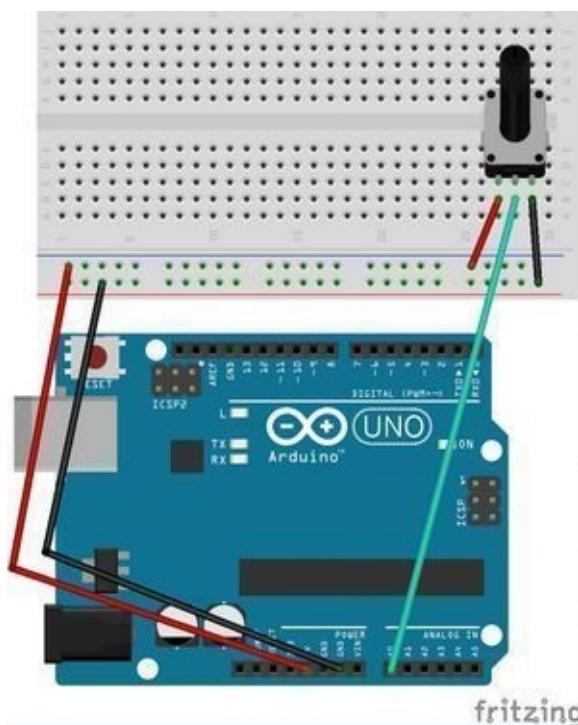
Objectif : contrôler un moteur de Poppy par un potentiomètre sur la platine Arduino.

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 V-REP.
- Exécutez les commandes python suivantes :

```
from poppy.creatures import PoppyHumanoid  
  
poppy = PoppyHumanoid(simulator='vrep', use_snap=True)
```

puis

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

Ouverture de Snap4Arduino

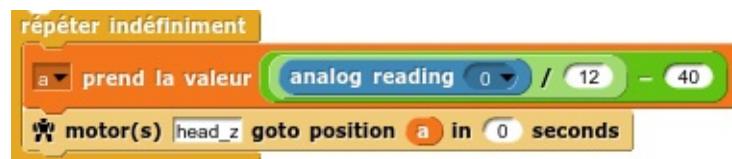
- Lancez Snap4Arduino et importez les blocs *pypyot-snap-blocks.xml*. (une fenêtre avertissant que le projet importé a été créé par Snap! apparaît ; elle est sans conséquence.)
- Dans les blocs Arduino, cliquez sur *Connect Arduino* pour établir la connexion entre Snap4Arduino et votre platine.



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

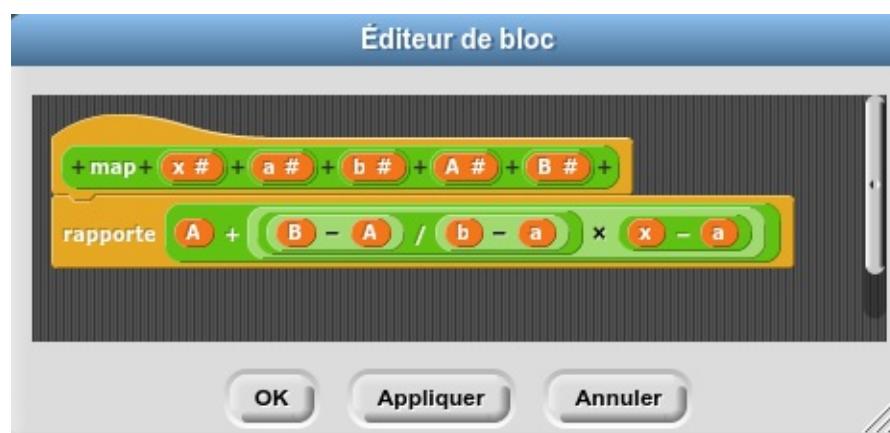
Commander un moteur via le potentiomètre

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° :

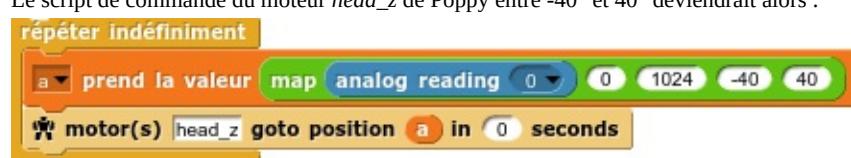


Remarques diverses

- 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.

Passer du robot Poppy simulé au robot physique

Un élément-clé du projet Poppy est de vous donner la possibilité de facilement passer d'un robot simulé (en utilisant V-REP par exemple) à un vrai robot. C'est assez utile lorsque vous :

- Développez une expérimentation dans laquelle vous pouvez tout configurer à partir de la simulation, pour la lancer ensuite sur le robot physique.
- Faites travailler vos élèves dans un premier temps sur des ordinateurs grâce à une simulation pour ensuite les faire essayer leurs travaux sur un robot physique partagé

Cependant, même si c'est fait de façon à ce que passage d'un plan à l'autre soit aisé, il y a quelques points à assimiler. Une des différences majeures est que lorsque vous travaillez via la simulation, tout se fait depuis votre ordinateur alors que quand vous utilisez un robot physique, le logiciel (par exemple les Python Notebooks) tout se fait depuis le robot.

À l'aide des notebooks Jupyter en Python

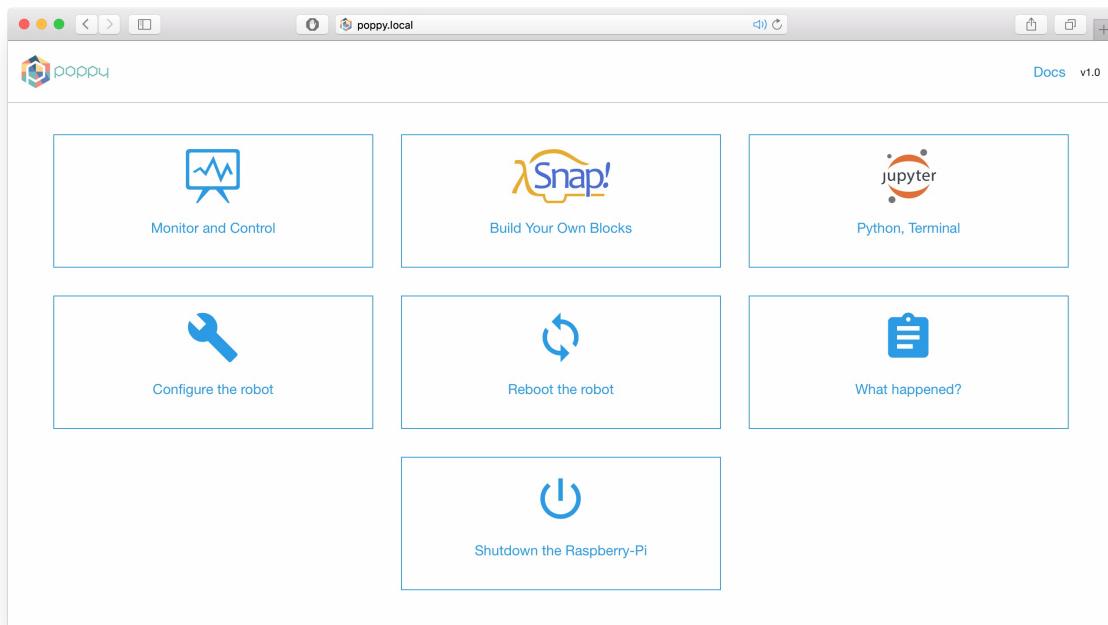
L'un des avantages de travailler avec les notebooks Jupyter est de pouvoir les utiliser dans une approche client/serveur. Chaque robot Poppy héberge un serveur Jupyter accessible via l'interface web (voir la [section démarrage rapide](#) pour plus de détails).

Lorsque vous travaillez via une simulation, tout est géré et stocké sur votre ordinateur. Lorsque vous travaillez avec un robot physique vous pouvez le programmer depuis un navigateur web sur votre propre ordinateur, mais vos notebooks Jupyter sont en fait stockés et exécutés dans le robot.

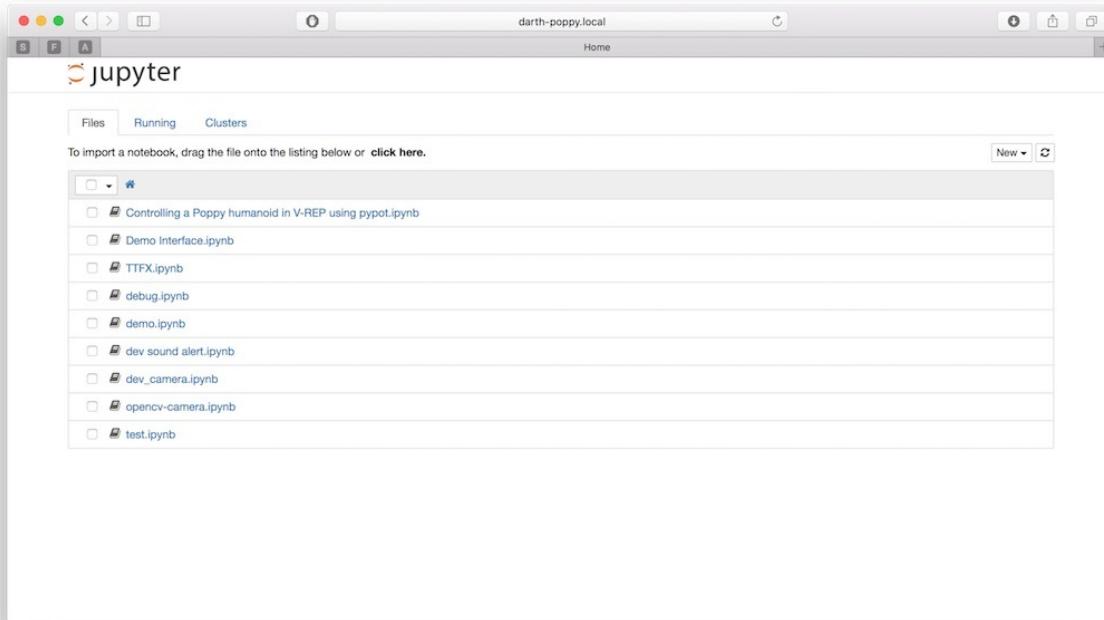
Donc pour passer de la simulation au robot physique, vous devez passer de votre instance locale Jupyter à l'instance hébergée par le robot. Les étapes sont décrites ci-dessous.

Se connecter à Jupyter sur le robot

Une fois connecté à l'interface web du robot <http://poppy.local> (nous supposerons ici que son nom d'hôte est *poppy*, remplacez-le par le nouveau nom d'hôte si vous l'avez changé), vous devriez voir un lien **ouvrir notebook Ipython**.



Lorsque vous cliquez dessus, Jupyter démarre sur le robot et vous redirige vers le serveur Web Jupyter. Vous devriez alors voir la racine du dossier notebook hébergé sur le robot :

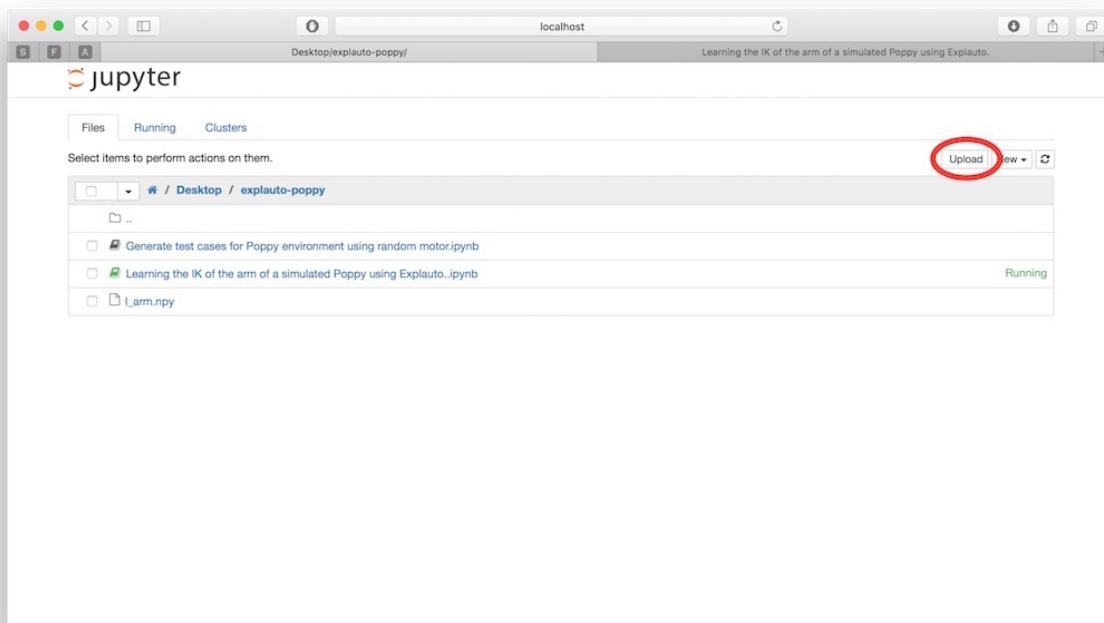


C'est ici que vous pouvez mettre vos propres notebooks. Bien sûr vous pouvez créer des dossiers, les organiser à souhait etc.

Note : Si vous avez besoin d'un accès plus précis ou d'une configuration plus avancée (telle qu'une autorisation), vous devez vous connecter au robot directement en utilisant SSH.

Télécharger un notebook

Une fois connecté au serveur Jupyter hébergé par le robot, vous pouvez directement utiliser l'interface Jupyter pour téléverser de nouveaux notebooks.



Le bouton rond permet de *télécharger* votre notebook local, stocké sur votre propre ordinateur, au robot. Ils peuvent alors être directement exécuté sur le robot.

Sachez que pour le moment, nous ne traitons pas avec des autorisations ou des sessions (comme JupyterHub par exemple), et donc toute personne ayant accès au robot peut utiliser ou supprimer tous notebooks stockés dans le robot.

Adapter votre code

Il y a peu d'endroits où vous devez réellement modifier votre code pour qu'il fonctionne avec un robot physique. Nous essayons de minimiser l'effort nécessaire dans la mesure du possible, cependant certaines étapes sont encore nécessaires.

Instanciation

Lorsque vous créez le robot, vous devez en fait préciser si vous êtes disposé à travailler avec un robot physique ou un robot simulé. Cela se fait simplement via un paramètre. Par exemple :

En travaillant avec V-REP :

```
from poppy.creatures import PoppyHumanoid

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

En travaillant avec le robot physique :

```
from poppy.creatures import PoppyHumanoid

poppy = PoppyHumanoid()
```

Bien évidemment, cela fonctionne pour tous les robots Poppy : Humanoid, Torso et Ergo Jr.

Il s'agit de la plupart des modifications que vous aurez à faire.

Les APIs spécifiques

Une partie de l'API dépend des plateformes. Par exemple, en utilisant V-REP vous avez accès aux fonctionnalités de *tracking* qui vous permet de récupérer n'importe quelle position 3D de l'objet. Mais cette méthode n'a pas d'équivalent dans le monde réel et n'est donc pas réalisable avec un robot physique.

Utiliser la propriété *simulée* est un bon entraînement si vous souhaitez écrire du code compatible pour les deux cas. Elle est automatiquement définie à la valeur correcte selon la façon dont vous instanciez votre robot. Par exemple,

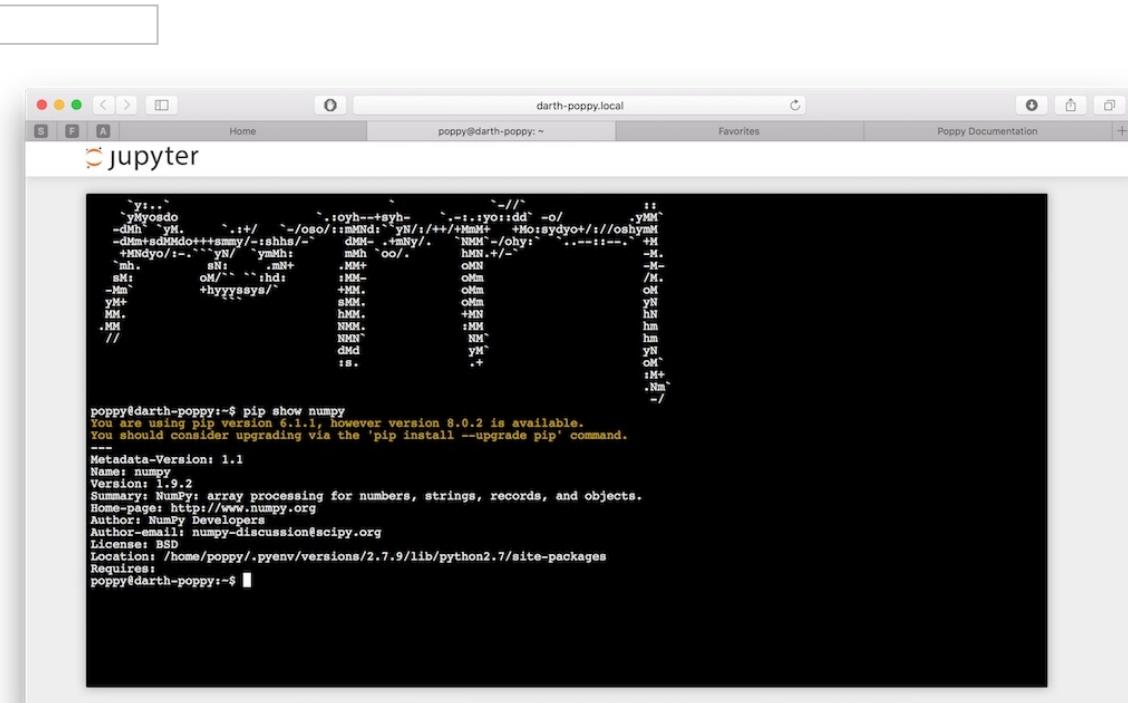
```
poppy = PoppyHumanoid(simulator='vrep')

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 et bibliothèques tierces

Le principal inconvénient de travail avec le mode client/serveur est que les versions de vos logiciels installés localement peuvent différer de celui installé sur le robot.

La version de Python installée sur le robot est Python 2.7 et dispose de la plupart des principales bibliothèques scientifiques (numpy, scipy, matplotlib, opencv). Une liste exhaustive des packages Python installés sera disponible bientôt <!--(TODO !)-->. Pour le moment, le moyen le plus simple d'y parvenir est d'utilisé un *terminal notebook* qui peut être exécuté directement depuis l'interface de Jupyter.



En utilisant la même technique, vous pouvez installer des bibliothèques tierces directement sur le robot. Les outils utilitaire [pip](#) et [conda](#) sont installés et devraient être utilisés lorsque cela est possible.

- Notez que la carte embarquée repose sur une architecture armv7 et donc certaines bibliothèques peuvent être difficiles à compiler. Nous maintenons une liste de recettes conda spécialement conçue pour cette plate-forme [ici](#).

Plus de contributions sont les bienvenues ! *

Avec Snap!

Cette page est encore vierge. Votre aide est nécessaire pour la remplir !

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 developing a new Poppy creature, it is first easier to simply define it in a configuration file or dictionary and instantiate 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 = 'min jerk'

        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 dependency 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 !

Réseau

Cette section contient une série d'astuces pour éviter les problèmes réseau.

Où est Poppy ?

Si vous recherchez votre robot sur votre réseau, rendez-vous au chapitre [zeroconf](#).

Contribuer au Projet Poppy

Si vous voulez faire partie du projet, la première étape est de devenir membre de la communauté sur le [forum Poppy](#). Le forum est l'espace consacré aux échanges entre les utilisateurs et les contributeurs. Vous êtes invités à partager votre projet et vos idées dans votre langue préférée.

Le projet Poppy étant un projet pluridisciplinaire, il y a plusieurs façons d'apporter sa contribution :

- En sciences de l'ingénierie telles que l'IA, l'informatique, la mécanique, l'électronique, apprentissage automatique...
- En sciences humaines, telles que les sciences cognitives, la psychologie...
- En sciences de la vie telles que la biologie, la biomécanique...
- Dans le domaine de la gestion de communauté, de la médiation scientifique, de la communication...
- En design, tel qu'en web design, design interface ou amélioration de l'expérience utilisateur...
- Des profils artistiques pour la conceptions d'animations qui recréent [une illusion de vie](#) et des émotions.

Si vous n'avez aucune idée de comment contribuer mais que vous en avez la volonté, nous vous invitons à **jeter un coup d'oeil aux "issues" ouvertes sur notre GitHub et aux appels à contribution**.

Pour les ninjas du GitHub, vous pouvez bien sûr créer des tickets pour signaler un problème ou développer de nouvelles fonctionnalités époustouflantes et ouvrir des *pull requests* pour demander à intégrer votre idée.