4 - ROS2 - Manipulation Cobot

- Universal Robot ROS2 Driver
- Commander un robot UR avec le driver ROS2
- Programmer un robot avec MoveIt2 - Jumeau Numérique
- Manipulation avancée avec AICA - déploiement Cloud
- Manipulation avancée avec AICA - déploiement on Premise

Universal Robot ROS2 Driver

Le Driver ROS2 pour les cobots Universal Robot a été développé en collaboration entre Universal Robots, le Forschungszentrum für Informatik (INRIA allemand) et PickNik Robotics. Plus précisément par nos voisins de Karlsruhe, en particulier Felix Exner (https://github.com/fmauch)). Le FZI est également à l'origine d'une proposition d'interface ROS standard pour les trajectoires Cartésiennes, qui est désormais implémentée dans les contrôleurs ROS Cartésiens d'Universal Robots. La proposition repose sur un état de l'art très intéressant des interfaces de programmation des marques de robot majeures.

Installation des paquets UR pour ROS2

- Réaliser l'installation de ROS2.
- Installer les paquets ros2 de Universal Robots

```
sudo apt install ros-humble-ur
```

- Installer rosdep pour installer automatiquement les paquets Debian dont dépendent les paquets ROS

```
sudo apt install python3-rosdep
```

- Si vous avez plusieurs versions de ROS2 installées, vérifiez que vous avez "sourcé" la bonne version de ROS2
- Mettre à jour les paquets dont ROS2 et les paquets UR dépendent

```
sudo rosdep init
rosdep update
sudo apt update
sudo apt dist-upgrade
```

Installation du simulateur hors-ligne URSim

https://github.com/UniversalRobots/Universal_Robots_ROS2_Driver#getting-started

https://github.com/UniversalRobots/Universal_Robots_ROS2_Driver#install-from-binary-packages

Le moyen le plus simple de découvrir et commencer à utiliser un robot UR avec ROS2 est d'utiliser la simulation de la console de programmation (teach panel) de son contrôleur. URSim est le simulateur hors-ligne de Universal Robots. Il permet de reproduire le comportement réel d'un robot quasiment à l'identique en se connectant à son adresse IP. Il est possible d'installer URSim 5 sur un Ubuntu <18 (non inclus!) ou dans une machine virtuelle (VirtualBox). Il faut se créer un compte pour télécharger le fichier URSim 5.13. Ubuntu 18 n'est plus supporté, et MoveIt2 tourne sur Ubuntu 22.

Sur Ubuntu 22

En attendant la sortie de URSim/Polyscope 6, **la manière la plus simple d'installer URSim 5 sur Ubuntu 22 est via conteneur Docker créé par UR Lab (pas maintenu officiellement)**.

Installer Docker

Depuis les dépôts officiels Ubuntu :

```
sudo apt update
sudo apt install docker-compose
```

Ajouter votre utilisateur au groupe docker afin de manipuler les containers sans avoir à utiliser sudo systématiquement :

```
sudo usermod -aG docker $USER
```

Fermer et rouvrir la session. Tester la bonne installation :

```
sudo service docker start
docker run hello-world
```

Installer le conteneur URSim

On suit le tutoriel fourni dans la documentation du driver UR ROS. Il existe une image docker fournie sur hub.docker.com, c'est très simple :

- Télécharger l'image docker

```
docker pull universalrobots/ursim_e-series
```

- Lancer l'image sur 192.168.56.101 avec l'URCap external_control préinstallé. Les programmes installés et les changements d'installation seront stockés de manière persistante dans `${HOME}/.ursim/programs`. Par exemple `-m ur5e`

```
ros2 run ur_robot_driver start_ursim.sh -m <ur_type>
```

- Ouvrir l'interface URSim en suivant les instructions qui s'affichent dans la fenêtre de terminal
  1. Voir URSim en utilisant une application VNC, en se connectant à `192.168.56.101:5900`
  2. Voir URSim en utilisant une application serveur web VNC

     http://192.168.56.101:6080/vnc.html

Sur Windows 10/11

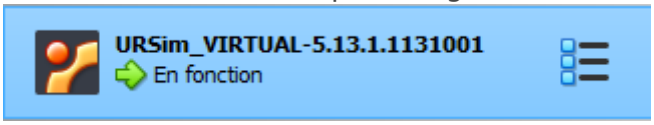
Pour les systèmes non-Linux, UR fournit une VM LUbuntu 14.04 qui peut tourner sous VirtualBox (gratuit) ou sur VMware (gratuit pour usage non commercial).

Installer VirtualBox

Télécharger et configurer la Machine Virtuelle

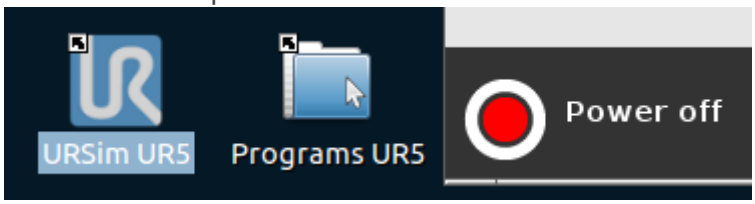
- Télécharger l'archive de la VM sur seafile ou en créant un compte chez UR
- Extraire l'archive dans `C:\Users\user\VirtualBox VMs\URSim_VIRTUAL-5.13.1.1131001`
- Dans l'interface VirtualBox cliquer sur `Ajouter`

- Sélectionner le Fichier machine virtuelle (`.vbox`) `C:\Users\user\VirtualBox VMs\URSim_VIRTUAL-5.13.1.1131001\URSim_VIRTUAL-5.13.1.1131001.vbox`
- La VM est entièrement préconfigurée, il suffit de la lancer

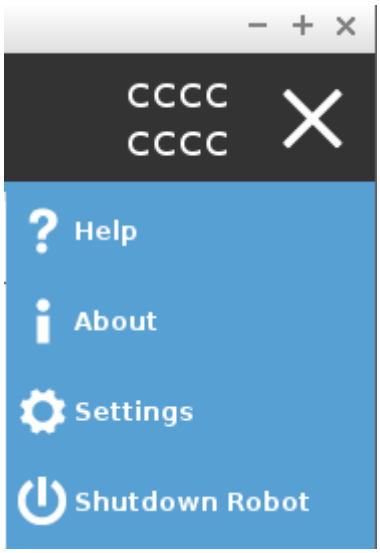


- La session Lubuntu est
  - user : `ur`
  - mdp : `easybot`

- Lancer URSim pour l'UR5e



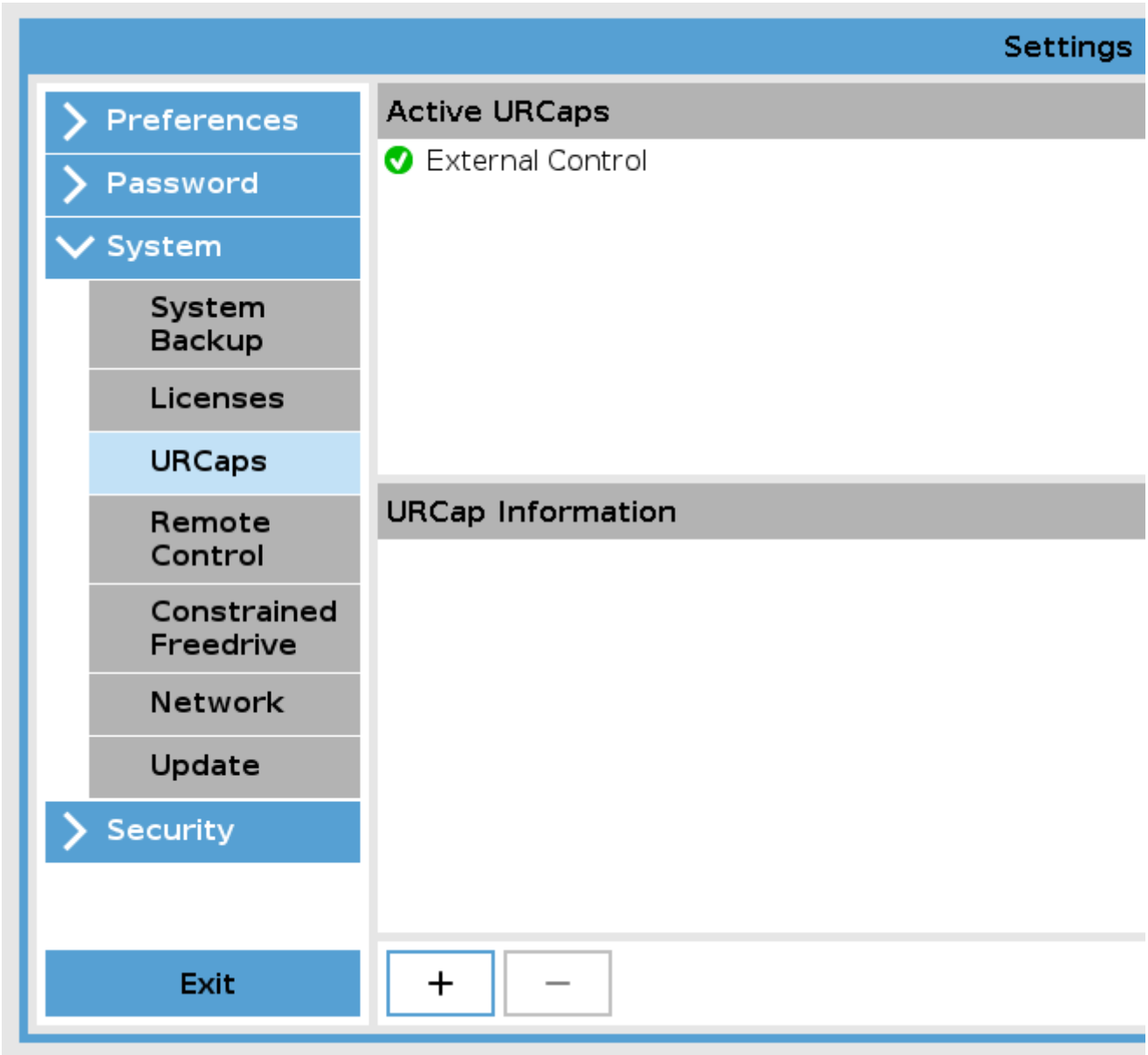
- Démarrer le robot en cliquant sur le bouton d'allumage "Power"
- Pour éteindre le robot



- Ou simplement éteindre la VM

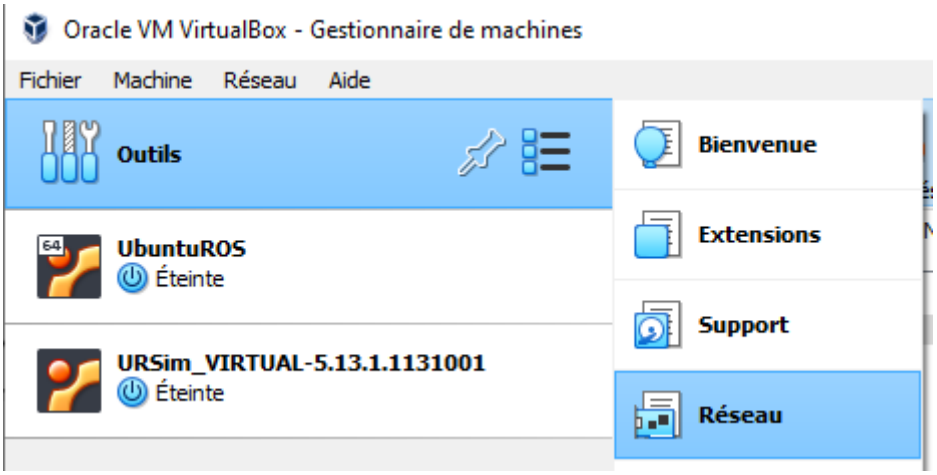
Ajouter l'URCap External Control

- Installer Terminator qui permet de faire des copier-coller
  `sudo apt install terminator`
- Se placer dans le dossier d'échange qui fonctionne comme une clé USB
  `cd /home/ur/ursim-current/programs.UR5`
- Lancer Terminator et télécharger le fichier `.urcap`
  `wget https://github.com/UniversalRobots/Universal_Robots_ExternalControl_URCap/releases/download/v1.0.5/externalcontrol-1.0.5.urcap`
- Démarrer URSim, ajouter l'URCap et redémarrer URSim

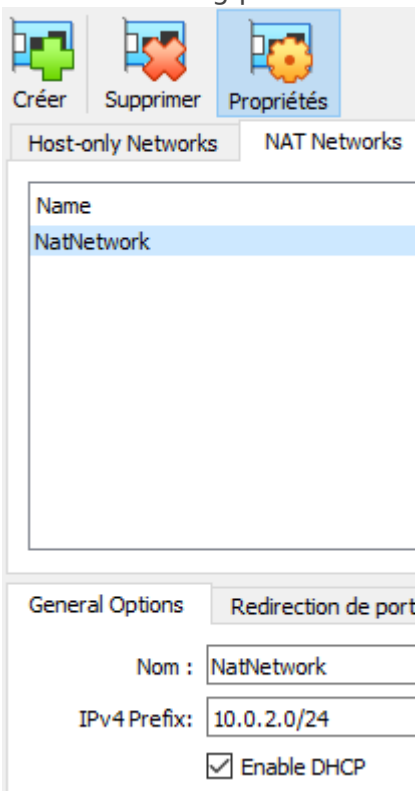


Configurer le réseau pour que les VMs communiquent

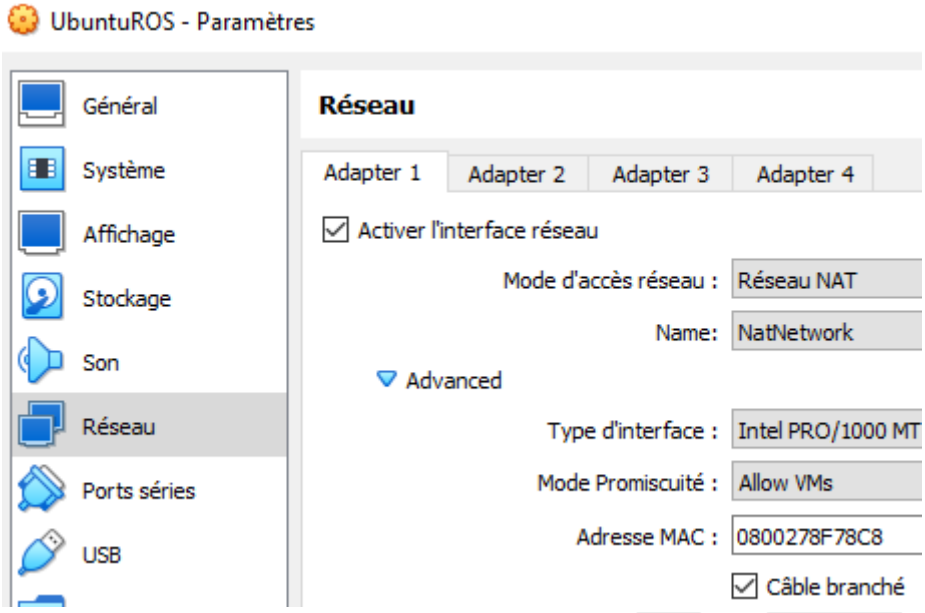
- Créer un réseau NAT dans les outils réseau



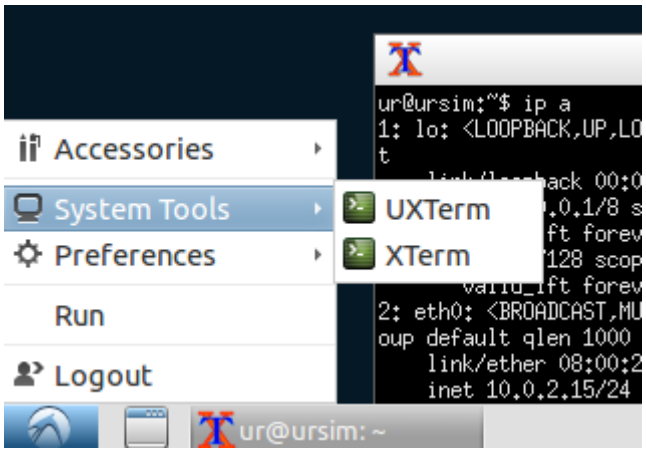
- Laisser la config par défaut



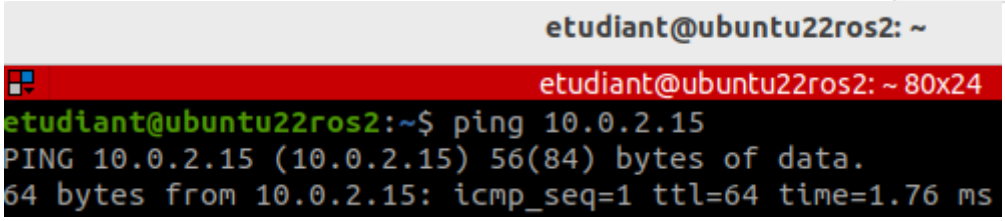
- Configurer chaque VM qui doit communiquer en `Réseau NAT`, par défaut elles seront en DHCP sur le réseau `10.0.2.0/24`



- Récupérer l'adresse IP de la VM URSim avec `ip a`, ici `10.0.2.15`



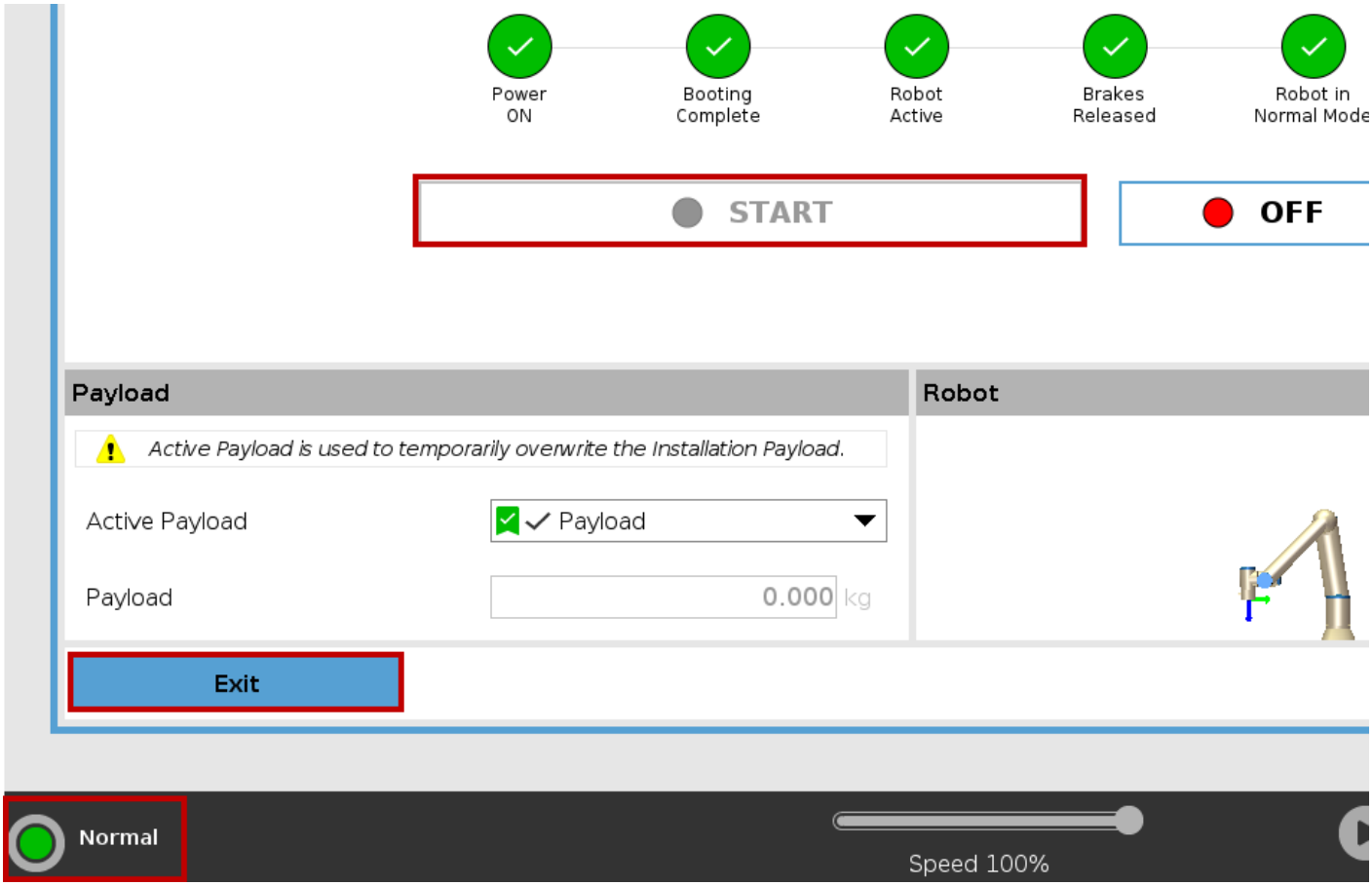
- Tester la communication de la VM ROS vers la VM URSim avec `ping 10.0.2.15`



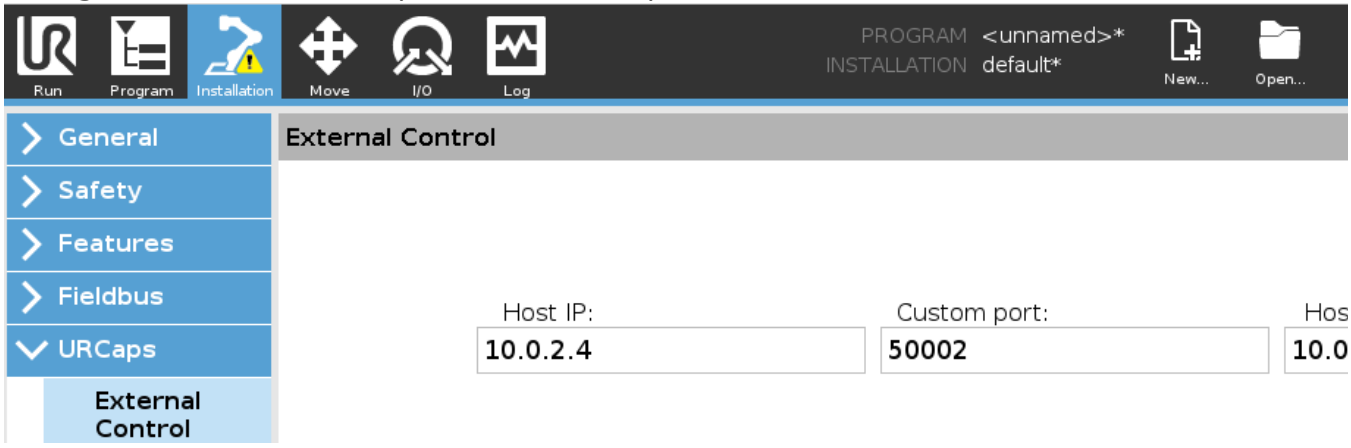
- Tester la communication de la VM URSim vers la VM ROS avec `ping 10.0.2.4`
- Tester la communication entre ROS et URSim, voir la section dédiée

Démarrage et configuration URSim

- démarrer le robot



- Configuer l'IP de la VM ROS pour l'autoriser à prendre le contrôle externe



-

Programmation hors-ligne avec URSim et MoveIt2/RViz

- Réaliser l'installation de ROS2.
- Installer rosdep pour installer automatiquement les paquets Debian dont dépendent les paquets ROS

```
sudo apt install python3-rosdep
```

- Si vous avez plusieurs versions de ROS2 installées, vérifiez que vous avez "sourcé" la bonne version de ROS2
- Mettre à jour les paquets dont ROS2 dépend

```
sudo rosdep init
rosdep update
sudo apt update
sudo apt dist-upgrade
```

- Installer colcon qui est le système de compilation de ROS2 avec mixin.

```
sudo apt install python3-colcon-common-extensions
sudo apt install python3-colcon-mixin
colcon mixin add default https://raw.githubusercontent.com/colcon/colcon-mixin-repository/master
colcon mixin update default
```

NOTES IMPORTANTES

- si vous utilisez Linux Mint 21.1 VERA (Mate ou Cinammon, basée sur Ubuntu 22 Jammy), il faudra toujours préciser la version d'Ubuntu `--os=ubuntu:jammy` dont on veut récupérer les paquets avec la commande `rosdep` :

```
sudo apt update && rosdep install -r --from-paths . --ignore-src --rosdistro $ROS_DISTRO -y --os=ubuntu:jammy
```

- Si vous travaillez dans une VM d'un ordinateur ayant moins de 20GO de RAM (WSL par exemple). Ou sur un ordinateur Ubuntu ayant moins de 10GO de RAM, il faudra lancer la compilation `colcon` sans parallélisation des tâches (1 paquet compilé à la fois) `--parallel-workers 1` :

```
colcon build --mixin release --parallel-workers 1
```

Déploiement du code source des Tutoriels de MoveIt

- Créer un répertoire de travail "workspace" pour la compilation du projet avec colcon

```
mkdir -p ~/ws_moveit2/src
```

- Se déplacer dans le workspace colcon et récupérer le code source des tutoriels MoveIt :

```
cd ~/ws_moveit2/src
git clone https://github.com/ros-planning/moveit2_tutorials -b humble --depth 1
```

Optionnel : Installation d'un environnement de développement MoveIt2 avec les dernières améliorations et résolutions de bug

Installation de MoveIt2 Humble sur Ubuntu 22.04 :

- Installer vcstool qui est un outil Python pour gérer les dépôts git composant un paquet ROS.

```
sudo apt install python3-vcstool
```

- Récupérer les autres dépôts git de MoveIt2 dont dépend moveit2_tutorials

```
cd ~/ws_moveit2/src
vcs import < moveit2_tutorials/moveit2_tutorials.repos
```

Note : si `vcs import` vous demande vos identifiants GitHub, tapez Entrer jusqu'à ce que ça continue. Pas besoin d'identifiant pour récupérer un dépôt GitHub public.

Installation des dépendances et compilation

- Installer les dépendances ROS2 Humble (paquets debian stables).

```
sudo apt update && rosdep install -r --from-paths . --ignore-src --rosdistro $ROS_DISTRO -y
```

- Compiler MoveIt Tutorials (20+ minutes si vous avez choisi de déployer l'environnement de développement)

```
cd ~/ws_moveit2
colcon build --mixin release
```

- Sourcer les paquets compilés :

```
cd ~/ws_moveit2
source ~/ws_moveit2/install/setup.bash
```

- Optionnel, si vous utilisez principalement ce workspace : Sourcer automatiquement le workspace colcon au lancement d'un Terminal

```
echo 'source ~/ws_moveit2/install/setup.bash' >> ~/.bashrc
```

Tester la communication entre ROS et URSim

https://docs.ros.org/en/ros2_packages/rolling/api/ur_robot_driver/usage.html#usage-with-official-ur-simulator

- Lancer URSim, par exemple avec un UR5e

```
ros2 run ur_robot_driver start_ursim.sh -m ur5e
```

- Ouvrir l'interface URSim dans le navigateur : http://192.168.56.101:6080/vnc.html --> cliquer sur Connect
- Faire tourner le driver UR ROS2

```
ros2 launch ur_robot_driver ur_control.launch.py ur_type:=ur5e robot_ip:=192.168.56.101
```

- Si vous bougez le robot dans URSim vous le verrez bouger dans RViz.

https://docs.ros.org/en/ros2_packages/rolling/api/ur_robot_driver/usage.html#example-commands-for-testing-the-driver

Envoyer des commandes au contrôleur

Avant d'envoyer des commandes, vérifier l'état des contrôleurs en utilisant `ros2 control list_controllers`

- Envoyer un objectif (goal) au contrôleur de trajectoire articulaire (Joint Trajectory Controller) en utilisant le noeud de démonstration du paquet ros2_control_demos. Dans nouveau Terminal (sans oublier de sourcer) :

  ```
  ros2 launch ur_robot_driver test_scaled_joint_trajectory_controller.launch.py
  ```

  Après quelques secondes le robot devrait bouger.

- Pour tester un autre contrôleur, il suffit de l'ajouter en argument de la commande `initial_joint_controller`, par exemple en utilisant joint_trajectory_controller :

```
ros2 launch ur_robot_driver ur_control.launch.py ur_type:=ur5e
robot_ip:=192.168.56.101 initial_joint_controller:=joint_trajectory_controller
launch_rviz:=true
```

- Et envoyer la commande avec le noeud de démo :

```
ros2 launch ur_robot_driver test_joint_trajectory_controller.launch.py
```

  Après quelques secondes le robot devrait bouger (ou sauter si la simulation est utilisée).

Auteur: Gauthier Hentz, sur le wiki de l'innovation de l'IUT de Haguenau

Commander un robot UR avec le driver ROS2

Installation d'Ubuntu avec des capacités temps-réel

Pour un usage basique, un Ubuntu (ou Linux Mint) classique permet de piloter le robot :

- Installer Ubuntu LTS 22.04
- Le noyau linux est capable de temps réel sans modification depuis sa version 6.12

Pour éviter des instabilités il est conseillé d'installer un noyau Linux avec des capacités temps-réel (PREEMPT_RT kernel). En particulier avec un robot de la Série-E, la fréquence de contrôle plus élevée peut entraîner des trajectoires non fluides sans noyau temps-réel.

- Vérifier qu'il reste au moins 25Go d'espace disque
- On se place dans un dossier pour la compilation

```
mkdir -p ~/rt_kernel_build
cd ~/rt_kernel_build
```

```
tar xf linux-4.14.139.tar
cd linux-4.14.139
xzcat ../patch-4.14.139-rt66.patch.xz | patch -p1
make oldconfig
```

Récupérer les sources du noyau temps-réel

- Regarder les versions LTS du noyau Linux : https://www.kernel.org/category/releases.html

- Regarder la version actuelle et les versions proposées par Ubuntu : Update Manager -> view -> Linux Kernel
- Regarder les versions maintenues activement du noyau PREEMPT_RT -> https://wiki.linuxfoundation.org/realtime/preempt_rt_versions
- Choisir la version du noyau Linux PREEMPT_RT maintenue activement correspondant à une version LTS et proposée par Ubuntu
- Exemple pour un Lenovo Thinkpad T480, Ubuntu 22.04
  - Actuellement installé : 5.19 (non-LTS)
  - Proposé : 5.19 et 5.15 (LTS)
  - On choisit 5.15 car LTS et activement maintenu en PREEMPT_RT
- Pour Ubuntu 24.04 : 6.1 (LTS) https://cdn.kernel.org/pub/linux/kernel/projects/rt/6.1/
- Récupérer les sources pour `5.15.107-rt62`

```
wget https://cdn.kernel.org/pub/linux/kernel/projects/rt/5.15/patch-5.15.107-rt62.patch.xz
wget https://cdn.kernel.org/pub/linux/kernel/projects/rt/5.15/patch-5.15.107-rt62.patch.sign
wget https://www.kernel.org/pub/linux/kernel/v5.x/linux-5.15.107.tar.xz
wget https://www.kernel.org/pub/linux/kernel/v5.x/linux-5.15.107.tar.sign
```

- Décompresser les fichiers téléchargés

```
xz -dk patch-5.15.107-rt62.patch.xz
xz -d linux-5.15.107.tar.xz
```

- Importer les clés publiques des développeurs du noyau et du patch (5.15-rt Joseph Salisbury 2026-10)

```
gpg2 --locate-keys torvalds@kernel.org gregkh@kernel.org
gpg2 --keyserver hkp://keys.gnupg.net --search-keys salisbury
```

- Vérifier l'intégrité des fichiers source

```
gpg2 --verify linux-5.15.107.tar.sign
```

> gpg: Good signature from "Greg Kroah-Hartman <gregkh@kernel.org>" [unknown]
> gpg: WARNING: This key is not certified with a trusted signature!
> gpg: There is no indication that the signature belongs to the owner.

```
gpg2 --verify patch-5.15.107-rt62.patch.sign
```

```
gpg: Good signature from "Tom Zanussi <tom.zanussi@linux.intel.com>"
[unknown]
gpg:                 aka "Tom Zanussi <zanussi@kernel.org>" [unknown]
gpg:                 aka "Tom Zanussi <tzanussi@gmail.com>" [unknown]
gpg: WARNING: This key is not certified with a trusted signature!
gpg:          There is no indication that the signature belongs to the owner.
```

Compiler le noyau temps réel

- Extraire l'archive tar, appliquer la patch et configurer le noyau temps-réel

```
tar xf linux-5.15.107.tar
cd linux-5.15.107
xzcat ../patch-5.15.107-rt62.patch.xz | patch -p1
make oldconfig
```

- Choisir l'option `4. Fully Preemptible Kernel (RT) (PREEMPT_RT_FULL) (NEW)` pour l'option `Preemption Model`

```
Preemption Model
  1. No Forced Preemption (Server) (PREEMPT_NONE)
> 2. Voluntary Kernel Preemption (Desktop) (PREEMPT_VOLUNTARY)
  3. Preemptible Kernel (Low-Latency Desktop) (PREEMPT)
  4. Fully Preemptible Kernel (RT) (PREEMPT_RT_FULL) (NEW)
choice[1-4]: 4
```

- Compiler le noyau

```
make -j `getconf _NPROCESSORS_ONLN` deb-pkg
```

- Après la compilation, installer les paquets `linux-headers` et `linux-image` dans le dossier parent (pas les paquets `-dbg`)

```
sudo apt install ../linux-headers-5.15.107-rt62_*.deb ../linux-image-5.15.107-rt62_*.deb
```

Définir les permissions utilisateur pour exécuter des tâches temps-réel

- Le Driver ROS2 va planifier des threads avec les permissions de votre utilisateur. Il faut donc autoriser votre utilisateur à utiliser lancer des threads avec une priorité temps-réel. On créé le groupe des utilisateurs temps-réel et on y ajoute l'utilisateur :

```
sudo groupadd realtime
sudo usermod -aG realtime $(whoami)
```

- S'assurer que `/etc/security/limits.conf` contient :

```
@realtime soft rtprio 99
@realtime soft priority 99
@realtime soft memlock 102400
@realtime hard rtprio 99
@realtime hard priority 99
@realtime hard memlock 102400
```

Note: Pour que ces changements soient pris en compte il faut se déconnecter et se reconncter. On redémarrera plus tard.

https://github.com/HowardWhile/Ubunutu22.04-RT-Kernel

Configurer GRUB pour toujours booter le noyau temps-réel

- Lister les noyaux disponibles

```
awk -F\' '/menuentry|submenu / {print $1 $2}' /boot/grub/grub.cfg
```

> menuentry Ubuntu
> submenu Advanced options for Ubuntu
>
> menuentry Ubuntu, with Linux 5.15.107-rt62
> menuentry Ubuntu, with Linux 5.15.107-rt62 (recovery mode)

- Définir le noyau `5.15.107-rt62` par défaut avec un pattern `"submenu_name>entry_name"`

```
sudo sed -i 's/^GRUB_DEFAULT=.*/GRUB_DEFAULT="Advanced options for Ubuntu>Ubuntu, with Linux
5.15.107-rt62"/' /etc/default/grub
$ sudo update-grub
```

Vérification de la capacité de préemption temps-réel

```
uname -v |  cut -d" " -f1-4
```

```
#1 SMP PREEMPT RT
```

Optionnel : Désactiver le CPU speed scaling

Les threads planifiés en temps-réel s'exécutent sans problème. Cependant, des composants externes tels que les systèmes de visual servoing, non planifiés pour le temps réel peuvent être interrompus par les fonctionnalités d'économie d'énergie des processeurs récents qui changent leur fréquence d'horloge de manière dynamique.

- Pour vérifier et modifier les modes d'économie d'énergie :

```
sudo apt install cpufrequtils
cpufreq-info
```

```
current CPU frequency is XXX MhZ
```

- Désactiver le changement de fréquence automatique :

```
sudo systemctl disable ondemand
sudo systemctl enable cpufrequtils
sudo sh -c 'echo "GOVERNOR=performance" > /etc/default/cpufrequtils'
sudo systemctl daemon-reload && sudo systemctl restart cpufrequtils
```

Configuration du robot UR

https://docs.ros.org/en/ros2_packages/rolling/api/ur_robot_driver/installation/robot_setup.html

Récupération de la calibration usine

Les robots sont calibrés en usine. Pour que les calculs cinématiques effectués dans ROS soient exacts, il faut récupérer les données de calibration. Sinon la précision des trajectoires envoyées depuis ROS et exécutées sur le robot risquent d'être de l'ordre du centimètre (au lieu du dixième de millimètre en temps normal).

https://docs.ros.org/en/ros2_packages/rolling/api/ur_robot_driver/installation/robot_setup.html#extract-calibration-information

---

Auteur: Gauthier Hentz, sur le wiki de l'innovation de l'IUT de Haguenau

Programmer un robot avec MoveIt2 - Jumeau Numérique

Prérequis : être arrivé au bout du tutoriel sur le Driver UR ROS2

Comment fonctionne la manipulation avec MoveIt ?

MoveIt2 est la plateforme de manipulation robotique pour ROS2. Il implémente un nombre important des dernières innovations en termes de :

- Planification de trajectoire
- Manipulation
- Perception 3D
- Cinématique
- Contrôle
- Navigation

https://moveit.picknik.ai/humble/doc/concepts/concepts.html

Premiers pas avec MoveIt dans RViz

Pour débuter avec MoveIt, on peut utiliser ses fonctionnalités de planification de trajectoire via le **plugin MoveIt Display** du logiciel de visualisation 3D de ROS **RViz**. C'est un outils très puissant pour débuguer des applications robotiques ROS. On verra que RViz est alors un jumeau numérique du vrai robot.

Les tutoriels pour débuter et approfondir ses compétences avec MoveIt sont en Anglais et fonctionnent avec le robot Panda de Franka Emika.

Nous reprenons ici le tutoriel pour débuter avec MoveIt, et l'appliquons à un UR5e avec le driver UR ROS2.

Avec un hardware simulé par ROS

Faire tourner le driver UR ROS2 :

1. Jusqu'à ROS2 **Humble**
   `ros2 launch ur_robot_driver ur_control.launch.py ur_type:=ur5e robot_ip:=yyy.yyy.yyy.yyy fake_hardware:=true launch_rviz:=false`
2. A partir de ROS2 **Jazzy**
   `ros2 launch ur_robot_driver ur_control.launch.py ur_type:=ur5e robot_ip:=yyy.yyy.yyy.yyy use_mock_hardware:=true launch_rviz:=false`

Avec la simulation URSim

Voir https://innovation.iha.unistra.fr/books/robotique-open-source/page/universal-robot-ros2-driver#bkmrk-installation-du-simu

Sous Ubuntu 22 - docker

- Démarrer la simulation URSim pour un UR5e

```
ros2 run ur_robot_driver start_ursim.sh -m ur5e
```

- Ouvrir l'interface URSim dans le navigateur : http://192.168.56.101:6080/vnc.html --> cliquer sur Connect
- Faire tourner le driver UR ROS2

```
ros2 launch ur_robot_driver ur_control.launch.py ur_type:=ur5e robot_ip:=192.168.56.101 launch_rviz:=false
```

Sous Windows - VirtualBox

- Télécharger la VM URSim avec External Control préinstallé
- Configurer le réseau NAT VirtualBox, récupérer les adresses IP avec `ip a` et tester la communication avec `ping 10.0.2.X`, cf. :
  - https://innovation.iha.unistra.fr/books/robotique-open-source/page/universal-robot-ros2-driver#bkmrk-configurer-le-r%C3%A9seau
- Démarrer URSim

- Démarrer le robot virtuel
- Tester la communication entre ros_control et l'URCap external control, cf. https://innovation.iha.unistra.fr/books/robotique-open-source/page/universal-robot-ros2-driver#bkmrk-https%3A%2F%2Fgithub.com%2Fr
- `ros2 launch ur_robot_driver ur_control.launch.py ur_type:=ur5e robot_ip:=10.0.2.5 initial_joint_controller:=joint_trajectory_controller launch_rviz:=true`

Avec le vrai robot

- Faire tourner le driver UR ROS2

```
ros2 launch ur_robot_driver ur_control.launch.py ur_type:=ur5e robot_ip:=192.168.0.10
launch_rviz:=false
```

Lancer MoveIt et RViz

```
ros2 launch ur_moveit_config ur_moveit.launch.py ur_type:=ur5e launch_rviz:=true
```

https://ur-documentation.readthedocs.io/en/latest/index.html ?

Déplacer le Robot avec le Plugin MoveIt dans RViz

- On veut lancer une requête de planification de trajectoire
- On utilise le plugin MotionPlanning qui permet de configurer la requête via une interface graphique

Résultat

ros2_moveit2_ros2control_commande_externet_UR5e.mp4

Planification de trajectoire avec OMPL

Ajouter un obstacle

- Choisir une configuration cible faisable, sans collision

ros2_moveit2_OMPL_RRTconnect_evitement_collision_automatique

Tester différents algorithmes d'OMPL

- Algorithmes RRT d'exploration rapide stochastique d'arbre
-

ros2_moveit2_OMPL_RRTstar_evitement_collision_longueur_optimisee

Résultat

En optimisant la trajectoire avec RRTstar, on obtient un mouvement fluide, qui évite les collision avec l'environnement et de longueur minimisée. Voir la vidéo réalisée en salle robotique de l'IUT.

Auteur: Gauthier Hentz, sur le wiki de l'innovation de l'IUT de Haguenau

Manipulation avancée avec AICA - déploiement Cloud

Prérequis

On déploie les TP avec une architecture client-serveur.

Avantages :

- Fonctionne sur postes Windows même peu puissants
- Pas besoin d'installer Docker, WSL, VirtualBox ou d'avoir de droits admin
- Déploiement d'autant d'environnements AICA qu'on veut sur une VM en un script

Côté serveur

- Une VM (machine virtuelle ou serveur physique) Linux
  - Avec un accès SSH pour un utilisateur dans le groupe `sudo`
  - De préférence avec GPU NVidia, les drivers propriétaires et `nvidia-container-toolkit` installés
  - Avec `docker.io` installé

Côté client

- Un PC avec une connexion rapide à la VM
  - Chrome/Chromium installé avec support de WebGL
  - VSCode/Codium installé avec l'extension `ms-vscode-remote.remote-ssh` (sur Codium tester `jeanp413.open-remote-ssh`)

> Si notre client est un PC Linux on peut avoir une expérience AICA complète en affichant RViz et AICA Launcher sur le PC : https://docs.aica.tech/docs/reference/manual-installation-launch#display-sharing

Installation de AICA sur le serveur via ssh

https://docs.aica.tech/docs/reference/manual-installation-launch

- Ouvrir VSCode
- Se connecter à la VM via SSH
- Ouvrir un Terminal dans VSCode
- Copier le dossier `docker` dans votre espace utilisateur `/home/user/`
- Copier la licence `aica-license-1.toml` dans `docker/aica/`
- Connecter la VM au dépôt Docker de AICA
  `cat /home/user/docker/aica/aica-license.toml | sudo docker login registry.licensing.aica.tech -u USERNAME --password-stdin`
- Build le ou les environnements docker nécessaires pour le TP depuis le fichier `launcher.toml` en leur donnant un nom, par ex. `aica-yolo-web` :

  ```
  cd docker/aica
  sudo docker build -f /home/user/docker/aica/aica-launcher-yolo-web.toml -t aica-yolo-web .
  ```

TP3 - Vision par IA avec Yolo

Pour déploiement sur un serveur avec GPU NVidia et `nvidia-container-toolkit` :

```
#syntax=ghcr.io/aica-technology/app-builder:v2

[core]
"image" = "v5.1.0"

[packages]
# add components
#"@aica/components/rl-policy-components" = "v2.0.0"
"@aica/components/advanced-perception" = "v1.0.0" # contains YoloExecutor
"@aica/components/core-vision" = "v1.1.2" # contains CameraStreamer
"@aica/foss/toolkits/ml" = "v1.0.0-cpu24.04" # prerequisite for YoloExecutor

# other extensions

# add hardware collections
```

Pour déploiement sur un serveur sans GPU :

```
#syntax=ghcr.io/aica-technology/app-builder:v2

[core]
```

```
"image" = "v5.1.0"

[packages]
# add components
#"@aica/components/rl-policy-components" = "v2.0.0"
"@aica/components/advanced-perception" = "v1.0.0" # contains YoloExecutor
"@aica/components/core-vision" = "v1.1.2" # contains CameraStreamer
"@aica/foss/toolkits/ml" = "v1.0.0-cpu24.04" # prerequisite for YoloExecutor

# other extensions
"@aica/foss/web-video-server" = "v0.1.0" # enables web streaming of video topics

# add hardware collections
```

Si elles ne sont pas déjà dispo dans `docker/` télécharger et compiler les composants dépendants, par ex. :

```
"my-local-package" = "docker-image://my-custom-component-package"
```

Démarrer une instance du conteneur `aica-yolo-web` nommée `aica` en lui passant le dossier persistant `aica-yolo-web/persistent/` qui sera disponible dans un dossier `persistent/` :

```
sudo docker run -it --rm --privileged --net=host -v /home/user/docker/aica/aica-license.toml:/license:ro -v /home/user/docker/aica/aica-yolo-web/persistent/:/persistent:rw -e AICA_SUPER_ADMIN_PASSWORD=12345678 --name aica aica-yolo-web
```

- L'argument `-e AICA_SUPER_ADMIN_PASSWORD=12345678` n'est nécessaire qu'au premier démarrage pour pouvoir créer un compte Admin.

Il peut y avoir des Warnings qui ne sont à ignorer et apparaissent si Cloud Storage n'est pas configuré ou si la vérification de license prend plus que quelques secondes :

```
[2024-11-18 13:38:16 +0000] [135] [INFO] Starting sync of cloud applications
[2024-11-18 13:38:16 +0000] [135] [WARNING] Sync failed
[2024-11-18 13:08:42 +0000] [151] [INFO] Waiting for licensing status... 5
[WARN] [1731935323.407252919] [EventEngine.ServiceHandler]: (404): Could not determine any license status
```

- Dans l'onglet `Ports` de VSCode, ajouter le port `8080`
- Dans le navigateur Chrome Ouvrir localhost:8080

- Se connecter avec les identifiants `super-admin`, `12345678`
- Créer un compte Admin (Profile > change password) et bien noter les identifiants.
- La configuration de cette instance AICA sera sauvegardée dans `aica.sqlite`, `aica.sqlite-shm` et `aica.sqlite-wal`
- Pour garder et dupliquer la config, on peut copier les fichiers `.sqlite` dans le dossier `persistent/`

Attacher un Terminal à l'intérieur l'instance `aica` de `aica-yolo-web` :

```
sudo docker container exec -it -u ros2 aica /bin/bash
ros2 node list
```

On voit bien que le contenu du dossier de la VM `aica-yolo-web/persistent/` est dispo dans le conteneur dans le dossier `persistent/` :

```
ls persistent
aica.sqlite aica.sqlite-shm aica.sqlite-wal
```

Détacher le Terminal du conteneur avec `CTRL+D` ou en tapant `exit`.

Stopper le conteneur : `docker container ps` puis `docker container stop <container_name>`.

Déroulé du TP3

Define a workcell setup
AICA studio -> Hardware -> modify URDF of e.g. UR5e

Build a custom package for Workcell or moveit python API script
https://github.com/aica-technology/community/blob/main/extensions/topic_based_ros2_control/aica-package.toml
source = "git://github.com/hellantos/ur5e_cell#start-of-training:ur5e_cell_description"

Manipulation avancée avec AICA - déploiement on Premise

Pour déploiement sur un PC Linux avec GPU et nvidia-container-toolkit :

```
#syntax=ghcr.io/aica-technology/app-builder:v2

[core]
"image" = "v5.0.1"

[packages]
# add components
#"@aica/components/rl-policy-components" = "v2.0.0"
"@aica/components/core-vision" = "v1.1.2"
"@aica/foss/toolkits/cuda" = "v1.0.0-cuda24.12"
"@aica/foss/toolkits/ml" = "v1.0.0-gpu24.12"

# add hardware collections
"@aica/collections/ur-collection" = "v4.2.0"
```

Pour déploiement sur un serveur sans accélération graphique :

```
#syntax=ghcr.io/aica-technology/app-builder:v2

[core]
"image" = "v5.0.1"

[packages]
# add components
#"@aica/components/rl-policy-components" = "v2.0.0"
```

```
  "@aica/components/core-vision" = "v1.1.2"
  "@aica/foss/toolkits/ml" = "v1.0.0-cpu24.04"

  # add hardware collections
  "@aica/collections/ur-collection" = "v4.2.0"
```

`cat /home/user/aica/aica-license.toml | sudo docker login registry.licensing.aica.tech -u USERNAME --password-stdin`

`sudo docker build -f /home/user/aica/aica-application.toml -t aica-runtime .`

`sudo docker run -it --rm --privileged --net=host -v /home/user/aica/aica-license.toml:/license:ro aica-runtime`