PROJ-H-402 - Computing Project: Extending ARGoS and TAM with Python

Alberto Parravicini

1 Introduction

ARGoS is a physics-based simulator for swarm robotics. Working with ARGoS requires the user to know either **C++** or **Lua**. The goal of this project was to provide an easy-to-use interface written in **Python**, a simple language often known also by people unfamiliar with programming.
TAM (Task Abstraction Module), a device used in swarm robotic, was also partially reimplemented in Python, and it is fully supported by the ARGoS simulator.

The following report is divided in two main sections, which describe the Python wrapper to ARGoS, and the TAM implementation, respectively.
For each section, it is provided an explanation of the design process, and of the structure of the code. Instruction to install the software are also provided.

2 A Python wrapper for ARGoS

2.1 *Introduction*

ARGoS is a complex physics-based simulator used in swarm robotics to model the behaviour and the actions of small robotic devices, such as `e-puck` or foot-bot. Each of these robots has a multitude of different sensors and actuators, that can be used to understand and interact with the surrounding environment or robots.

Working with real robots can be expensive and time-consuming, and therefore a simulator is an essential tool in the development of most applications and researches.
Even more so, a simulator can be an invaluable resource for students wishing to learn more about swarm robotics.

As a consequence, it is clear the importance of having a simulator which is not only powerful, but also very easy to use.
Swarm robots usually follow very simple instructions and patterns, and there are obvious advantages in being able to implement such tasks in a fast and error-free way.

A simulation in ARGoS is composed by three parts:

- A **configuration** file, written in an **xml**-like format, which specifies the entities (the robots, and other objects) that are part of the simulation. This file also loads the required controllers and loops functions (see below), and other options used by the simulation.
- One or more **controllers**, which describe the behaviour of the robots. Controllers have a set of predefined abstract functions that have to be implemented by the user, in order to specify the actions of the robots at the start, at the end, and at each step of the simulation.
- A **loop function**, if necessary, that will customize the events at the start or end of each simulation step.

ARGoS is implemented in **C++**, and uses **C++** as the main choice for writing simulations. Even though this language is known to most people working in robotics, it is still rather cumbersome to use in practice, due to its inherent complexity, and the need to recompile the code at each small update.
Moreover, it can be argued that the performances offered by **C++** can hardly be beneficial when the tasks to run are relatively simple to execute, as it is often the case for swarm robots.

ARGoS also offers a wrapper written in **Lua**, which allows to write simulations in a faster and simpler way.
Still, **Lua** is still not a very well known language: most developers, especially students, probably won't be able to use this wrapper without spending additional time to learn the language.
This wrapper doesn't have all the features of the original **C++** implementation, but it still allows to control the most important actuators and sensors of the robots.

It looks natural at this point to look for a different solution, which can provide the accessibility of the **Lua** wrapper, without sacrificing any functionality.
This is the idea behind the **Python** wrapper for ARGoS: a wrapper which is developed to be as similar as possible to the **Lua** version, and that offers the same functionalities, if not more.

2.2 *Installation*

This section will describe how to install ARGoS and the **Python** wrapper, and how to run simulations with it. Specific details about the structure of the simulations are left to the later sections.

The wrapper was mainly tested on **Windows 10**, using the **Linux subsystem** (basically, a shell for **Ubuntu 16.04**). In order to install it, it is recommended to follow the guide at http://www.terriblesysadmin.com/?p=76.
The wrapper was also briefly tested on a native **Archlinux** distribution.

2.3 *Installing ARGoS*

Here it is described how to install the basic ARGoS simulator.
This guide is adapted from the one found at https://github.com/ilpincy/argos3.

- Download the ARGoS source code at https://github.com/ilpincy/argos3.
- (Alternatively, clone the repository).
  ```
  git clone https://github.com/ilpincy/argos3.git argos3
  ```
- Install the dependencies.
  ```
  sudo apt-get install libfreeimage-dev libfreeimageplus-dev
  qt5-default freeglut3-dev libxi-dev libxmu-dev liblua5.2-dev
  lua5.2 doxygen graphviz graphviz-dev asciidoc
  ```
- Move into the repository folder, called `argos3`.

- Create a folder `build_simulator`.
  ```
  mkdir build_simulator
  ```
- Move into the folder `build_simulator`.
  ```
  cd build_simulator
  ```
- Run *cmake*
  ```
  cmake ../src
  ```
- Run *make*
  ```
  make
  ```
- Compile the documentation (required!).
  ```
  make doc
  ```
- Install ARGoS.
  ```
  sudo make install
  ```
- Make sure that ARGoS is installed system-wide.
  This step is taken from https://lonesysadmin.net/2013/02/22/error-while-loading-shared-libraries-cannot-open-shared-object-file/
  Make sure that the dynamic linker checks `/usr/local/lib`:
  ```
  cat /etc/ld.so.conf
  include ld.so.conf.d/*.conf
  /usr/local/lib
  ```
- Update the cache.
  ```
  sudo ldconfig
  ```
- It is possible to check if ARGoS runs correctly by running the examples found at https://github.com/ilpincy/argos3-examples

2.4 *Installing e-puck*

The **Python** wrapper fully support **e-puck**, one of the robots that can be used in ARGoS. This section details how to install the **e-puck** plugin.
More details can be found at
https://github.com/lgarattoni/argos3-epuck/blob/master/doc/Installation.pdf

- Clone the repository the **e-puck** plugin repository.
  ```
  git clone https://github.com/lgarattoni/argos3-epuck.git argos3-epuck
  ```
- Move into the repository folder, `argos3-epuck`.
- Create a `build` directory, move into it.
  ```
  mkdir build
  cd build
  ```
- Install **e-puck**
  ```
  cmake ../src
  make   sudo make install
  ```
- Check that **e-puck** appears among the available entities in ARGoS. `argos3 -q entities`
- Optionally, try some of the examples found in the previous folder, as there are also examples that support **e-puck**.

2.5 *Installing the Python wrapper*

Finally, it is possible to install the **Python** wrapper. Note that it requires to install **Boost** (more details on this are given later).
The wrapper was tested with **Boost 1.61.0**, but this guide should apply also to more recent versions of the library.

This section is based on http://www.boost.org/doc/libs/1_61_0/more/getting_started/unix-variants.html and on https://eb2.co/blog/2012/03/building-boost-python-for-python-3-2/, using **Python 3.5** instead of **3.2**.

- Download **Boost** at http://www.boost.org/users/history/version_1_61_0.html
- Move in the folder where **Boost** was downloaded, and move it to `/usr/local/`.
  ```
  mv boost_1_61_0.tar.bz2 /usr/local
  ```
- Extract **Boost** in the same location.
  ```
  tar -bzip2 -xf boost_1_61_0.tar.bz2
  ```

- Install the *developer* version of Python.
  ```
  sudo apt-get install python3-dev
  ```
- Move to the **Boost** folder.
  ```
  cd /usr/local/boost_1_61_0/
  ```
- Build **Boost**.
  ```
  ./bootstrap.sh -with-python=python3.5
  ./b2
  sudo ./b2 install
  ```
- Clone the **Python** wrapper repository.
  ```
  git clone https://github.com/KenN7/argos-python argos-python
  ```
- Move into the repository folder, create a `build` folder and move into it.
  ```
  mkdir build
  cd build
  ```
- Build the wrapper.
  ```
  cmake ../src
  make
  ```
- From the build folder, it is possible to launch some examples to see if the wrapper is working correctly. For instance:
  ```
  argos3 -c ../examples/aggregation_10.argos
  ```

2.6 *Technical details of the implementation*

2.7 *Description of the wrapper components*