HOWTO - Start with the Architecture

Related courses: Robotica, PAS1 and PAS2

Team BORG

February 25, 2013

Contents

1 Introduction

This document introduces you to the BORG architecture and lets you get started with your first behavior tree using the Nao. Please also have a look at the “System Architecture” presentation for a short overview. Refer to the “Howto Repository” document for more information on getting access the the software[1]. It is recommended to first read section 9 if you are following the PAS course.

Please note that this document is a work in progress and will be updated in time.

[1]This might not be required for the PAS course; download the tar.gz from the PAS website.

2 Preparation

1. Make sure you cloned the repository (see the document "HOWTO - Use the repository")[2].
2. Make sure the following lines are added to your `.bashrc` file (located in your home[3] directory):

   ```
   export BORG=$HOME/sudo
   export PYTHONPATH=/opt/choregraphe/lib:$PYTHONPATH
   export PYTHONPATH=$BORG/brain/src:$PYTHONPATH
   export LD_LIBRARY_PATH=/opt/choregraphe/lib:$LD_LIBRARY_PATH
   export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$BORG/brain/src/util
   ```

   The variable `BORG` should refer to the location of the root directory of the (cloned) repository.
3. Also add the following line to your `.bashrc` file in case you are working on the AI systems (using your own AI account):

   ```
   source /home/student/vakken/Robotica/sdk/set_env_vars
   ```

4. Also, if you are experiencing errors related to missing python libraries, try adding the following `PYTHONPATH` to your `.bashrc` file:

   ```
   export PYTHONPATH=$BORG/brain/src:$PYTHONPATH
   ```

3 Installation of Dependencies

If you are *not* using the standard AI or Ubuntu systems in the lab, you need to install a lot of libraries yourself. The following instructions may be incomplete, deprecated, not work on your system, or you are required to use different packages or versions.

1. Follow the instructions as indicated in section 2.
2. Follow the instructions as indicated in section 9.
3. Add the "lib" directory of the choregraphe package to your `PYTHONPATH` and `LD_LIBRARY_PATH`:
   (a) Open your "`~/.bashrc`" file.
   (b) Add the following at the end (modify the path) (should be on one line!):

   ```
   export PYTHONPATH=/home/ron/doc/rug/choregraphe-suite-1.12.5-linux64/
   lib:$PYTHONPATH
   ```

   (c) Add the following at the end (modify the path) (should be on one line!):

   ```
   export LD_LIBRARY_PATH=/home/ron/doc/rug/choregraphe-suite-1.12.5-linux64/
   lib:$LD_LIBRARY_PATH
   ```

   (d) Reload it for every terminal that you have already opened (or just restart the terminal):

   ```
   # source ~/.bashrc
   ```

4. Install specific python libraries:

   ```
   # sudo apt-get install python-scipy python-pygame python-cv
   ```

[2]Or download and unpack the `borg_src_**-**-**.zip/tar.gz` from: `http://www.ai.rug.nl/crl/uploads`
[3]By default "/home/student" for the PAS course.

5. For using the pioneer, install Aria:
   `http://robots.mobilerobots.com/wiki/ARIA` (Simply download/unpack the “*.tgz” and execute `sudo make install`.)

6. You can find and install remaining dependencies as follows (for Ubuntu or Debian):

   (a) Search for packages:

   ```
   # apt-cache search <word_1> <word_2> <etc.>
   ```

   (b) Install packages, type;

   ```
   # sudo apt-get install <package_name>
   ```

   (c) or you can use the GUI to install packages:

   ```
   # synaptic
   ```

4 Starting the architecture

You can start the architecture (including the brain, communicators and modules) either by using the `start.sh` or the `startGUI.sh` script.

4.1 Using the commandline

The following instructions will make you start up the system (the example behavior and example vision module) from scratch. This is recommended to do so first, so you know what is actually happening on the background.

1. Make sure to generate all behaviors if you haven’t does so already:

   ```
   # cd $BORG/brain/src/
   # python basebehavior/generate_behaviors.py
   ```

2. Start the communicator in another terminal on each system (for the PAS course you only need to start it once)[4]:

   ```
   # cd $BORG/brain/src/
   # python communicator/communicator.py
   ```

3. Start the motionController on port 12345 (*not* required for the PAS course)[5]:

   ```
   # cd $BORG/brain/src/pioneercontrol/pioneercontroller/
   # ./motionController 12345
   ```

4. Navigate to the `brain/src/` directory and start the architecture using the `start.sh` script:

   ```
   # cd $BORG/brain/src/
   # LAPTOPS="none" ./start.sh config/config_example
   ```

   By setting “`LAPTOPS="none"`” you force the system to not start anything automatically.

5. If everything is working correctly, you should now see that the different example behaviors are run in sequence together with the dummy example module.

[4]The communicator is used on each system to automatically start or stop vision modules such as the blob detector.

[5]The motionController is used to control the pioneer mobile platform. You might first need to build an actual working controller by executing the “make” command in the `$BORG/brain/src/pioneercontrol/pioneercontroller/` directory.

6. Press Ctrl-C to shutdown the BORG system in the same terminal in which you started the "`start.sh`" file.

7. To be sure that everything is shutdown, you may want to use the following command:

   ```
   killall python -9
   ```

8. And if you are also using speech recognition:

   ```
   killall python -9 || killall java -9
   ```

4.2 Using the GUI

1. Navigate to the `brain/src/` directory.

2. Start the GUI tool using the `startGUI.sh` script:

   ```
   # ./startGUI.sh
   ```

3. Open the configuration window by click on the "Setup" option.

4. Specify the configuration file to use under "Configuration File".

5. Specify whether or not you are using the nao and/or the pioneer.

6. Save the option by pressing "Ok" (the options will be saved once you closed the GUI).

7. Start the brain (and thus the behaviors, and possibly the communicators and related modules) by pressing the "Start Brain" button.

5 Main Configuration File

An example configuration called "`config_example`" is located under "`brain/src/config`".

1. At the top, the starting behavior and its arguments (a dictionary) can be specified, for example:

   ```
   [general]
   starting_behavior = balltrack({})
   ```

   Please refer to the "Behavior Architecture" document for more information on passing arguments to behaviors.

2. Specify under `[body]` the number of robots for each type (e.d. Nao, Pioneer, Quadcopter) and its IP address and port number.

3. Perception/vision modules to use and their arguments can be specified under `modules_settings` under section `vision_controller`, for example:

   ```
   modules_settings:
       obstacledetectormodule = obstacledetector video_source=kinect
   ```

   The first part specifies the name of the instance, the second (after =) the name of the module and the remainder is used to specify the arguments.

4. After specifing the (instances of the) modules, one can specify on what host the instance has to run, for example:

   ```
   modules:
       localhost = obstacledetectormodule
   ```

In this case module the instance `obstacledetectormodule` is run on the same host as the Brain (e.d. `localhost`).

Please refer to the document "The Vision Controller" for more information about the vision controller, communicators, modules and its configuration.

6 The Behavior Architecture

Please refer to the document "Behavior Architecture" for more information.

6.1 Creating a (first) behavior

Follow the following guidelines to create a (first) behavior:

1. Edit "src/config/behaviors_config" and add a new behavior (for example "my_new_behavior"). Also, define the post condition (simply specify "False" if you do not care) and possibly exceptions (not mandatory).
2. Generate all behaviors using the **"python basebehavior/generate_behaviors.py"** command.
3. Copy the generated template "behaviors/my_new_behavior/my_new_behavior_x.py" to "behaviors/my_new_behavior/my_new_behavior_0.py".
4. Edit the new file and add your sub-behaviors (and its preconditions) in the **implementation_init()** function.
5. Add the actual behavior code to the **implementation_update()** function.
6. Create your own config file in the "src/config" directory.

6.2 A more complex behavior tree

Follow the following guidelines to create a more complex behavior tree:

1. Define the hardware that you require (Nao, Pioneer, etc.)
2. Define the sensors that you require (e.d. Kinect, Webcam, Laser Range Finder, etc.).
3. Define the computers and its modules that you are going to use (obstacle avoidance, object recognizers, etc.). Have a look in the `vision` directory for all available modules.
4. Create all behaviors (low-leven, high-level).
5. Create a new configuration file and add all relevant information.

Low-level behaviors generally contain more code in the `implementation_update` function (controlling specific movements and such of the robot). More high-level behaviors contain more code in the `implementation_init` function; stating the the subbehaviors it uses and its preconditions.

Keep in mind that all update functions (e.d. your implementation of the `implementation_update` function), are run in sequence with the (maximum) frequency as specified in the main configuration file. So if one of those update functions block (for example, by a call to the `time.sleep()` function), the whole behavior tree stalls.

7 Using the Memory

The singleton memory module is used to communication and store information between behaviors and modules. Please refer to the "The Memory" for more information.

8 Logging Facilities

It is recommended to use the internal logging facilities of the architecture. This allows you to either increase or decrease the verbosity of the system without having to comment or uncomment print statements. Please refer to the "Logging Facilities" document for more information.

9 The Nao (and related software)

9.1 Using the Nao

Please adhere the following guidelines:

1. **You should always be prepared to catch the robot!**
2. Deenslave the robot if you are not using it temporarly (this prevents the motors from heating up).
3. Turn of the robot if you are not using it (this prevents the CPU from heating up).
4. Always put the robot back in the correct box together with the power supply.

9.2 Choregraphe

9.2.1 AI Systems

On the AI systems, you can start Choregraphe simply by typing `choregraphe` in the command line. If this does not work, you can try to run Choregraphe using one of the binaries located under `/home/student/vakken/Robotica/sdk`, for example:

```
# /home/student/vakken/Robotica/choregraphe-suite-1.12/bin/choregraphe-bin
```

If your are having trouble finding NaoQi, you can also try to add one of the libraries paths under `/home/student/vakken/Robotica/sdk` in your `.bashrc` file, for example:

```
export PYTHONPATH=/home/student/vakken/Robotica/choregraphe-suite-1.12/lib:$PYTHONPATH
```

9.2.2 On your own computer

You can run Choregraphe on your own computer by downloading it from Aldebaran users website (see section 9.5). Simply download the archive, unpack it and set `PATH` and `PYTHONPATH` in a similar as been specified in section 9.2.1

For more information on using choregraphe:
`http://www.aldebaran-robotics.com/documentation/index.html`

9.3 Accessing the Python SDK

You can use the buildin "`src/body/nao.py`" module in the BORG system to access the Nao Python SDK[6].

In order to control the Nao robot within a behavior, make sure to:

1. Specify the amount of Nao robots to use and their IP addresses in your config file (see section 5).
2. Now you can access each Nao robot using the `nao(...)` function:

   ```
   self.nao = self.body.nao(0)
   ```

3. The standard library as provided by the BORG system has a few standard functionalities such as for example;

[6]See for more information: `http://www.aldebaran-robotics.com/documentation/dev/python/index.html`

(a) making the robot speak;

```
self.body.nao(0).say("Hello␣everybody!")
```

(b) making the robot move to a specific location (1, 0.5) with an angle of 0.1 radians while stopping if an object is detected within 25cm;

```
self.body.nao(0).moveNav(1, 0.5, 0.1, 0.25):
```

(c) and making the robot execute a specific choregraphe behavior:

```
self.body.nao(0).start_behavior("sitdown")
```

Please refer to the actual "`src/body/nao.py`" module for more information.

To access all functionalities of the Nao robot, you need to communicate with the robot using so called proxies. You can access these proxies using the "`get_proxy(...)`" function. For example to retreive the "ALMotion" proxy and make the robot move its head up, you can do the following:

```
self.body.nao(0).get_proxy("motion").setStiffnesses("Head", 1.0)
self.body.nao(0).get_proxy("motion").angleInterpolation( \
        "HeadPitch", 25 * almath.TO_RAD, 1.0, True)
```

Using the "`get_proxy(...)`" function, you can access the following proxies: "motion, navigation, tts, video, frame, memory, vision, balltracker, leds". Refer to the link stated above for more information on these proxies.

9.4 Recording and using your own Choregraphe behaviors

1. Of course you can create your own behavior in Choregraphe using a subset of the existing behaviors (by tying the up together and saving the project).
2. However, you can also record your own movement using the Animation box:
   `http://www.aldebaran-robotics.com/documentation/software/choregraphe/animation_mode.html`
3. Once you created and saved your project (make sure the final box is fully connected on both ends), you can transfer the behavior to the robot:
   `http://www.aldebaran-robotics.com/documentation/software/choregraphe/panels/behavior_manager_panel.html?highlight=behavior%20manager`
4. You can also use the behavior manager (see previous link) to start and stop all behaviors that are currently located on the robots.
5. Once the behavior is transfered to the robot, you can execute it with the BORG architecture from your behavior in the following way:

   ```
   self.nao = self.body.nao(0)
   self.nao.start_behavior('introducemyself_bas')
   ```

   Have a look in the `brain/src/body/nao.py` file for more information.

Tips and guidelines:

1. It is generally a good idea to record the movement slowly and then increase its framerate in order to increase its speeds.
2. It is also a good idea to split up your recordings in separate boxes. This will allow you to easily tie them up together and create different behaviors with different speeds without having to record the whole movement over and over again.

3. Make sure your final behavior in Choregrahe is connected on both ends (so the program knows when the behavior is supposed to start and end).

4. Use existing Choregraphe behaviors to make the nao sit and stand (your can also use the behavior manager for this (see one of the previous links)).

5. Only record those parts of the Nao body that you are actually going to use. Otherwise you will record the whole body which may result in improper and dangerous behavior.

6. **Be careful;** if you connect the boxes in Choregraphe in a recurrent way, multiple signals may be created which might result in multiple behaviors being executed. For more information:
`http://www.aldebaran-robotics.com/documentation/software/choregraphe/tutos/boxes.html?highlight=wait%20signal`

7. **Be careful;** always make sure you execute your behavior from a correct and known position (the robot might otherwise tip over!).

9.5 Resources

1. Publicly available documentation:
`http://www.aldebaran-robotics.com/documentation/index.html`

2. You can find more resources such as the latest binaries and documentation of the Nao on:
`http://users.aldebaran-robotics.com`
Username: `homerug`
Password: ******* (Ask for it.)

10 Standards

Please refer to the "Standards" document for more information on standards and guidelines related to coding and creating documentation.