UPennalizers RoboCup 2010 Open Source Release

University of Pennsylvania

September, 2010

1 Introduction

This code release has support for both Linux and MacOS.

2 Getting Started

In order to get the software to work on your machine there are a few packages that you must download and install first.

- Download and install Lua 5.1
  - You can download Lua here: `http://www.lua.org`
- Download and install Boost C++ Libraries. This will depend on which version of the code you will use:
  - `Nao`
    If you are going to be using the Nao, download the latest SDK provided by Aldebaran through their download center. Create a link in `/usr/local/include/` to the boost folder in the SDK. This will also work with the Webots simulation if you wish to use it.
  - `LC2 and Webots` If you are going to be using the LC2 or just the Webots simulation (and not the Nao), download the latest version of Boost here:
    `http://www.boost.org` Create a link to the Boost header files `boost_1_43_0/boost` (your version may differ) in the `/usr/local/include/` directory.

2.1 Setting Up Webots

If you are interested in using the Webots Nao Simulator then you must also do the following.

- Download and install Webots

  - You can download Webots for Linux or MacOS here: `http://www.cyberbotics.com`
  - Install Webots in the `/usr/local/` directory.
  - Create a link to the `webots` executable in `/usr/local/bin/`
  - NOTE: You do not need a license to run our code release. The demo version is all that is required.

- Create a link to the `WebotsController` we provided in the `webots/projects/contests/robotstadium/controllers/` directory:
  - First create a backup of the original `nao_team_0` directory.
  - Create a link to the `WebotsController` directory named `nao_team_0`
  - Do the same for `nao_team_1` if you want the code to be used for both teams, otherwise it will run the default controller.

3 Code Structure

The code is divided into two main components: high and low level processing. The low level code is mainly written in C/C++ and compiled into libraries that have Lua interfaces. These libraries are used mainly for device drivers and anything designed to execute quickly (e.g. image processing and forward/inverse kinematics calculations). The high level code is mainly scripted Lua. The high level code includes the robots behavioral state machines which use the low level libraries. The following will contain a brief description of the code following the provided directory structure. The low level code is rooted at the Lib directory and the high level code is rooted at the Player directory.

3.1 Low Level (`./Lib`)

`Platform`
: There are three directories contained here, one corresponding to the three robot platforms supported. The code contained in these directories is everything that is platform dependent. This includes the robots forward/inverse kinematics and device drivers for controlling the robots sensors and actuators. All of the libraries have the same interface to allow you to just drop in the needed libraries/Lua files without changing the high level behavioral code. The directory trees for each platform (Webots/Nao/LC2) are the same:

  `Body`
  : Body contains the device interface for controlling the robot's sensors and actuators. This includes controlling joint angles, reading IMU data, etc.

`Camera`
: Camera contains the device interface for controlling the robots camera.

`Kinematics`
: Kinematics contains library for computing the forward and inverse kinematics of the robot.

`ImageProc`
: This directory contains all of the image processing libraries written in C/C++: Segmentation and finding connected components.

`Util`
: These are all of the C/C++ utility function libraries.

`CArray`
: CArray allows access to C arrays in Lua.

`Shm`
: Shm is the Lua interface to Boost shared memory objects (only used for the Nao).

`Unix`
: This library provides a Lua interface to a number of important Unix functions; including time, sleep, and chdir to name a few.

`NaoQi`
: This contains the custom NaoQi module allowing access to the Nao device communication manager in Lua.

3.2 High Level (`./Player`)

`player.m` is the main entry point for the code and contains the robots main loop.

`BodyFSM`
: The state machine definition and states for the robot body are found here. These robot states include spinning to look for the ball, walking toward the ball and kicking the ball when positioned.

`Config`
: This directory contains the only high level platform depended code. They are in the form of configuration files. There is one configuration file per platform. The walk parameters, camera parameters and the names of the device interface libraries to use are all defined in the configuratoin files.

`Dev`
: This directory contains the Lua modules that to import for controlling the devices (actuators/sensors) on the robot. jitem `Data`
This directory contains any logging information produced. Currently that is only in the form of saved images.

`HeadFSM`
: The head state machine definition and states are located here. The head is controlled separately from the rest of the body and transitions between searching for the ball and tracking it once found.

`Lib`
: Lib contains all of the compiled, low level C libraries and Lua files that were created in Lib.

`Motion`
: Here is where all of the robots motions are defined. It contains the walk engine along with keyframe motions used for the get-up routine and kicks.

`Util`
: Utility functions are located here. The base finite state machine description and a Lua vector class are defined here.

`Vision`
: The main image processing pipeline is located here. It uses the output from the low level image processing to detect objects of intereset (mainly the ball).

`World`
: This is the code relating to the robots world model.

4 Compiling

There are three phases for getting the code running. The first is compiling all the necessary C/C++ libraries and the NaoQi module (if needed). The second is 'setup', which consists of copying the necessary low level libraries and Lua files from the `./Lib` directory to the `./Player/Lib` directory so they can be used by the high level code. The final set is installing. This is only necessary for the Nao and LC2 (not currently supported but coming soon). If you are only using the Webots simulator you will not have to install anything. We provide Makefiles and scripts to complete all of these tasks assuming you have all external dependencies installed correctly.

`Webots`
: Use the following command from the `./Lib` directory to setup Webots:
`$ make setup_webots`

`Nao`

1. Download the SDK, CTC and the latest opennao OS image tool from the download center provided by Aldebaran.
2. Shut off the Nao and remove the headpiece. Take the USB drive from the robot and mount it on your computer.

3. Use the `flash-usbstick` tool provided in the Aldebaran SDK to install a clean version of the OS onto the USB drive.
4. Once complete, replace the USB drive in the robot. Start the robot and wait for it to completely boot. Then shut the robot down and remove the USB drive.
5. Remount the USB drive on your computer and note the path to the USB root partition and the user partition on the drive.
6. From the `./Install` directory, run the `install_nao.sh` script:
   `$ ./install_nao <path/to/usb/root/partition> <path/to/usb/user/partition>`
   The paths should look something like `/media/<USB Name>/`
7. After the script completes place the USB drive back into the robot. Replace the headpiece and turn the robot on.
8. The robot will boot and start running the Player code. It will start in the bodyIdle state (walking in place). Press the chest button to transition into the normal body state machine (searching for the ball, kicking, etc). You can press it again to go back into idle.

`LC2`

The LC2 is not currently supported but will be coming soon...