Robocup Tutorial

Mohsen Kaboli, Zhiliang Wu

September 22, 2017

1 Building the working environment

1.1 System

The working environment is **Ubuntu 14.04**, which is already installed on all the computers and laptops in the lab. If it's your first time to work with it, please get in contact with the supervisors and we will provide some useful tutorials for help.

1.2 Prerequisite for building the project

To build the project, some software are required. More details are discussed in [1](cf. 2.3.4).
To sum up, it could be done with following commands:

```
sudo apt-get install qt4-dev-tools libglew-dev libxml2-dev clang graphviz
```

1.3 Downloading the package

Go to the Github repository of B-Human team and download the required packages. Please take care that we use **coderelease2015**.
In addition, 32 bit version of *Naoqi* (*C++ SDK 2.1.4 Linux 32*) should be downloaded and installed with the command:

```
cd <BHuman-CodeRelease>/Install
./installAlcommon.sh <pathto32bitC++NAOQi.zip>
```

1.4 Building the project

The whole project can be built with:

```
cd <BHuman-CodeRelease>/Make/Linux
make all
```

Also, different components can be built seperately, which are:

- bush
- Controller
- libbhuman
- libgamectrl
- libqxt
- Nao
- qtpropertybrowser
- SimRobot
- SimRobotCore2
- SimRobotEditor

- SimRobotHelp
- SimulatedNao

All components canbe built in three configurations, *Release, Develop* and *Debug*. The default seting is *Develop* like:

```
cd <BHuman-CodeRelease>/Make/Linux
make <component> [CONFIG=<configuration>]
```

For more details about the description of these components see [1](cf. 2.2)

2 (Option) Flashing the Robot

This part is provided for a new robot. Although all robots have already been flashed, it is meaningful to do everything from scratch again to understand the whole procedure.

2.1 Preparing the USB and flashing

The atom system image and corresponding *Flasher* should be downloaded from `https://community.aldebaran.com`. Then the following steps should be done:

- Extract the *Flasher* and execute

```
cd <Flasher>
./flasher
```

  Factor reset should be chosen before refleshing.

- Plug the USB to the backside of NAO and press the chest button until the chest turns blue(around 5 seconds).
- The installation process will start and be monitored by the LED lights around NAO's ears.
- When finished, the chest button should be pressed and NAO will tell its initial IP-Address. Note down the IP-Address for future use.

More details for the flashing procedure can be found at `http://doc.aldebaran.com/2-1/nao/boot_process_nao.html#upgrading-process-nao`.

2.2 Create the robot

Each robot has its unique IP-Address, which looks like $AAA.BBB.CCC.DDD$ (IPv4 IP-Address). Here, we set $AAA = 169, BBB = 254$ for all robots' wired configuration (For wireless settings, we set $AAA = 10, BBB = 0$). And CCC is for the team ID and DDD for robot ID. For example, 169.254.29.173 is the robot 173 from team 29. In addition, we should also assign a nickname to it such as *Neue*.
Before creating the robot, configure $AAA.BBB$ in template files *wireless* amd *wired* in *Install/Network* and *default* in *Install/Network/Profiles*. Then the new robot can be created like:

```
cd <BHuman-CodeRelease>/Install
./createRobot -t CCC -r DDD <nickname>
./addRobotIds -ip <Initial IP-Address> <nickname>
```

2.3 Conguration of the Network

The robot can only be connected with cables after reflashing. Connect the robot with the computer and add a *Ethernet Connection*. Set the *method* in *IPv4 Settings* as *link-local only*. Now the computer could connect with the robot. Try

```
ssh nao@<Initial IP-Address>
```

to check the reachablity of the robot. The IP-Address is the one given by Nao after reflashing.

2.4 Copy the compiled code to robots

After connecting to the robot successfully, we can install the robot and copy the complied code to it. To do this, it can be done with

```
cd <BHuman-CodeRelease>/Install
./installRobot <initial IP-Address>
```

After this step, the robot IP-Address will be changed into $AAA.BBB.CCC.DDD$.
Reconnet the robot with *ssh* and copy the compiled code to robot with:

```
cd <BHuman-CodeRelease>/Make/Linux
./copyfiles Develop AAA.BBB.CCC.DDD -r
```

More options for the command $copyfiles$ can be found in [1](cf. 2.5)

2.5 Setting up the wireless configuration

To set up the wireless configuration, the files in *Install/Network/Profiles* should be configured. Create a new *NAOwifi.wpa* file, the syntax can be referred from the *example.wpa* file. One example looks like

```
ctrl_interface=/var/run/wpa_supplicant
ctrl_interface_group=0
ap_scan=1

network={
  ssid="NAOSWIFI"
  key_mgmt=WPA-PSK
  pairwise=TKIP
  group=TKIP
  psk="naoapp99"
}
```

Then the wireless profiles can be updated with

```
cd <BHuman-CodeRelease>/Make/Linux
./copyfiles Develop AAA.BBB.CCC.DDD -w NAOwifi.wpa
```

Plug out the cable and connect to the WIFI (*NAOSWIFI*) with the password *naoapp99*. Now you should be able to connect with the robot through *ssh* command.

3 Connecting to Nao and playing around in SimRobot

3.1 Connecting to Nao

After previous setting-up steps, Nao can be connected with both cable and WIFI. The reachability can be checked with *ssh* command in the terminal. After successful connection, some commands can be executed directly in the terminal to control the robot.

- **nao start|stop|restart** starts, stops or restarts NAOqi.
- **bhumand start|stop|restart** starts, stops or restarts the bhuman executable.
- **status** shows the status of NAOqi and bhuman.
- **stop** stops running instances of NAOqi and bhuman.
- **halt** shuts down the NAO.
- **reboot** reboots the NAO.

3.2 Playing around in SimRobot

Before executing the *SimRobot*, it is recommended to connect to the robot through *ssh* command first to check the reachability of the robot. In addition, *status* command can be used to check both *Naoqi* and *bhuman* are running. If one of them are not running, give *naoqi/bhuman* to run these two executables in the robots.
SimRobot is an executable created during 1.4. It locates in *Build/Linux/<conguration >/SimRobot*.
The B-Human Code Release 2015 contains basic playing environments and configurations that can be used right from the start. Look at */Config/Scenes* and you will see *.ros2* files which describe the game modalities such as the number of players and starting positions. The corresponding *.con* files are used for example for playing field configurations.

3.2.1 Testing real robots using *RemoteRobot.ros2*

RemoteRobot is used to connect to a real robot and observe all its sensor data in real time. It is not a simulation, but can be used to check calibration, vision and behaviour.

- use camera views to check for example if ball-, goal- or obstacle-detection work.
- observe the real robots behavior and the states. Check if state transitions work e.g. after having detected the ball (State searchForBall), the robot should turn towards the ball (State turnToBall).

3.2.2 Executing commands

Open the *console* (double- click on it), and type a command

```
get representation:MotionRequest
```

You'll get a long string back, in which you can change some settings. For example, *motion = specialAction* means that the robot will execute a specialAction command, which is described a bit further as 'specialAction = playDead'. You can see other options by changing it to some false value, and pressing return. The options for specialAction are 'playDead', 'sitDown', 'stand', 'standHigh'. So if you change *specialAction* to *stand* will make the robot stand up. Other possibilities can be found from trial and error.
After setting up the real field, this *SimRobot* will become more powerful as it will be used for calibration and monitoring the state of the robot, which will be discussed in the following section.

4 Calibration in real-field

Calibration is very important for the robot. Each time before the game, the calibration should be implemented according to the field and the luminance environment. The calibration process can be done in the exactable *SimRobot*.

4.1 Joint Calibration

In order to ensure a proper performance of the robot's motion modules, its joints have to be calibrated correctly. Therefore a joint calibration can be done with the help of the simulation environment SimRobot. For an easy calibration, the following steps must be followed:

1. Bring the robot into a standing position.
2. Call the Joint Calibrator in SimRobot:

```
call JointCalibrator
```

3. The actual joint offsets will be displayed in SimRobot's console and it is now possible to adjust them directly in the console.
4. Adjust the offsets until the considered joints are in the desired position. For an easy calibration process, we can recommend the following calibration steps:
   (a) Get feet into a planar position.
   (b) Get feet into a parallel position (Feet should be 10cm apart from each other, measured from the grey plastic covers of the ankle-joints).
   (c) Get body into an orthogonal upright position (Torso must have no x/y rotation).
5. Save the adjusted offsets with the help of the following command:

```
save representation:CameraCalibration
```

   To make sure the calibration file is really changed according, *ll* command can be used to check the status of the calibration configuratino file(in *Config/Robot/Default/*).
6. Finally copy the new calibration to the robots by using the copyfiles-script:

```
cd <BHuman-root-directory>/Make/Linux
./copyfiles <Develop/Debug/Release> <IP_AAA.BBB.CCC.DDD>
```

4.2 Color and Camera Calibration

In order to ensure that the robots' perception modules can work properly, a prior color and camera calibration is required. Both calibrations can be done with the help of the simulation environment SimRobot.

4.2.1 Color Calibration

1. Perform a calibration of the white color by using the provided Auto-White-Balance function. Therefore, use the following console commands in SimRobot:

```
dr module:CameraProvider:DoWhiteBalanceUpper
dr module:CameraProvider:DoWhiteBalanceLower
```

2. Open the following information views: "segmented lower" and "segmented upper" and "ColorCalibration" (If these information views are not present in SimRobot, it is necessary to create those first.)

```
vi image ImageLower
vi image upper ImageUpper
```

3. Adjust the provided sliders in the ColorCalibration information view until the segmented images show good results for the chosen color (See Figure 1 for an exemplary calibration of the green color).

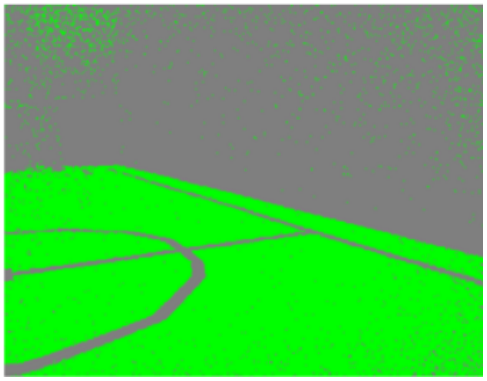

Figure 1: Exemplary configuration of the green color

4. Right-click into the ColorCalibration information view and press "save calibration".
5. Repeat this procedure for every color. One thing to metion is that the color of the ball is **orange**, rather than **red**.
6. Finally, copy the new calibration to the robots by using the copyfiles-script:

```
cd <BHuman-root-directory>/Make/Linux
./copyfiles <Develop/Debug/Release> <IP_AAA.BBB.CCC.DDD>
```

4.2.2 Camera Calibration

There are three modes in Camera Calibration, namely automatic, default and manual. The following shows the process of automatic camera calibration.

1. Start the automatic camera calibration module:

```
call CameraCalibrators/Automatic
```

2. Place the robot at a defined point on the field (e.g. Centerpoint (0,0)) and specify this spot in SimRobot with the following command:

```
set module:AutomaticCameraCalibrator:robotPose rotation= 0deg; translation = {x= 0; y=0;}
```

3. Start the calibration process:

```
dr module:AutomaticCameraCalibrator:start
```

4. Start optimization process:

```
dr module:AutomaticCameraCalibrator:optimize
```

5. Check if the created drawings of the field lines match the real field lines, if not, go back to point 3 and restart the calibration process.
6. Save the new calibration:

```
save representation:CameraCalibration
```

7. Finally copy the new calibration to the robots by using the copyfiles-script:

```
cd <BHuman-root-directory>/Make/Linux
./copyfiles <Develop/Debug/Release> <IP_AAA.BBB.CCC.DDD>
```

However, it may finish the calibration with large deviation to the groundtruth. So the default and manual Calibration are recommended if there are large deviation. They can be implemented in the console in SimRobot by

```
call CameraCalibrators/Default
```

Then the shortcut for different functions will shown in the console. The default calibration will calculate the result for the upper and lower camera according to the point chosen by us. The view of the camera can be changed by clicking on the screen and keep pressing *shift*. But it seems that it is not very necessary to do it because usually this leads to the failure of convergence. What important for the point collection is that the point should be collected both in upper camera and in lower camera, so that it will converge faster and easier.
If the Default Calibration is not satisfying, the manual calibration can be used with

```
call CameraCalibrators/Manual
```

to finely tune the parameter. The direction of different axis can be known from trial and error. Figure 2 shows an example of the camera Calibration.

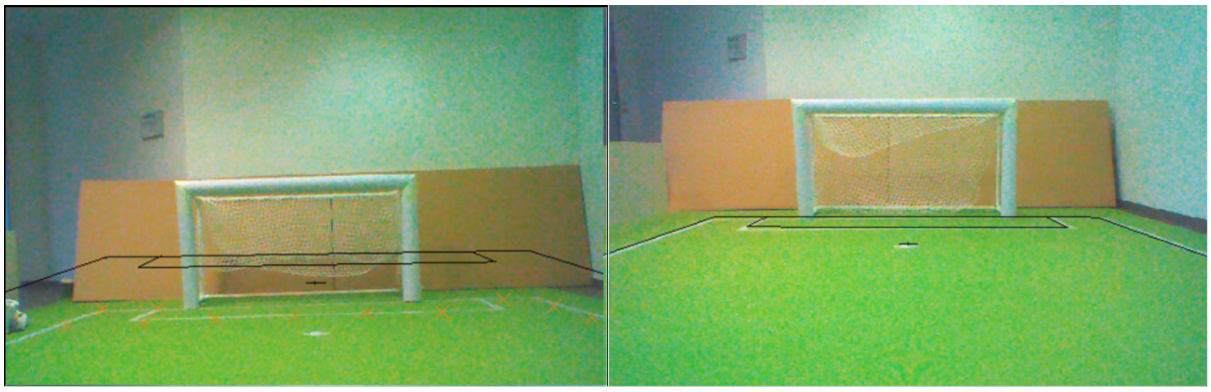

Figure 2: Calibration: Before vs After

4.3 Playing a game

After the previous steps, the robots should be able to play a real game in principle.
Do it and you will see many problems.
It's time to fix these problems by yourselves and start coding.
Some issues are opened in *Gitlab* project page as mini-projects for the improvement.
(See `https://gitlab.lrz.de/ga35goz/RoboCup/issues`)

References

[1] Thomas Röfer and Tim Laue and Jesse Richter-Klug and Maik Schünemann and Jonas Stiensmeier and Andreas Stolpmann and Alexander Stöwing and Felix Thielke, *B-Human Team Report and Code Release 2015*.Only available online: `http://www.b-human.de/downloads/publications/2015/CodeRelease2015.pdf`