AI Project

Kicking a street hockeyball

Arnoud Visser,
Bardia Khalesi(6108741), Ravi Kumar Gupta(6163548)

15 December 2009

Abstract

The aim of this project is to improve the ball kicking modules of *Nao* robot used in the RoboCup **Standard Platform League**. Since this year the Technical Committee has decided to use a new ball, a *street hockey ball* which is smaller and heavier than the ball used in the previous years, we have tried to design a more effective behavior of kicking the ball. To do this, we used *Choregraphe* simulator along with the existing code of *B-Human team*, and modified the special actions module related to kicking. In this manner, we designed a stable gesture of kicking which provides the robot with capability of kicking the ball faster and stronger. In the future, our work can be extended using machine learning techniques like reinforcement learning.

Introduction

Every year there is an international competition called "Robocup", in which machines (either real robots or virtual robots in simulation) compete with each other in different fields by playing different games like soccer, or rescue. The primary idea behind the competition is to make a team of robots which can play against football champions human team in the year 2050. This motivates scholars around the world to improve the related robotics' fields[3]. The Standard Platform League is a RoboCup robot soccer league, in which the robots operate completely autonomously, and all the teams use identical robots[5]. Since 2009, the official robot of the Standard Platform League's competition is Nao, a humanoid manufactured by *Aldebaran* in Paris, France. In addition, the ball used in our project is an orange no bounce Stag hockey balls which is 60 gram and a diameter of 68.5 millimeter (but the official ball is 65mm in diameter and it weighs 55g). Due to its heavier weight and smaller size compared to the previous Robocup official ball (Aibo ball), this ball is easier to handle.

In this matter, this project was defined to design a kick motion in a way that robot would be in a stable and balance position to kick the ball faster and stronger. Therefore, inasmuch as the code of the last year champion team was available as an open source code on the Internet and included an efficient basic modules like walking and perception along with an operative simulator, we would prefer to employ this simulator to create a more effective kicking movement suited to the size of the new ball. However, since behaviors such as kicking was considered as special actions and had been hard coded, we exploited Choregraphe, first, to assess efficacious values of the joints' motions of the robot in any steps of kicking a ball and then integrated the result with the code.

In this paper, primarily, we are going to introduce the Nao robot. In the next step, we will speak about the Choregraphe and B-Human team's code in detail as the basis of our work. Then, we will elaborate our implementation; further results and considerations are thereafter discussed in conclusion.

1 Nao

Nao is a 60 cm high robot and entirely programmable. The system running in Nao is an operating system based on Linux, but it can be programmed by different programming languages on different platforms (Windows, MacOS, etc.) via cross-platform tools such as Choregraphe, Gostai, etc. Choregraphe includes all graphic interfaces, behavior libraries needed to create movements; Gostai is developing Urbi, and also fully interfaced with C++, Java and Matlab[4].

The Nao robot has 26 (Figure 1) joints that make the robot be able to pose in any desirable positions. In addition, it has an inertial central unit consists of an accelerometer (3 axes) and a gyrometer (2 axes), and pressure sensors in its soles that provide the robot with data by which it can realize if it is standing up or lying down on its back or front. This helps the robot to know how it should move to stand up or lie down. Furthermore, the robot is equipped by two pairs of Ultra-sound senders/receivers that enable the robot not to bump into any obstacle[4].

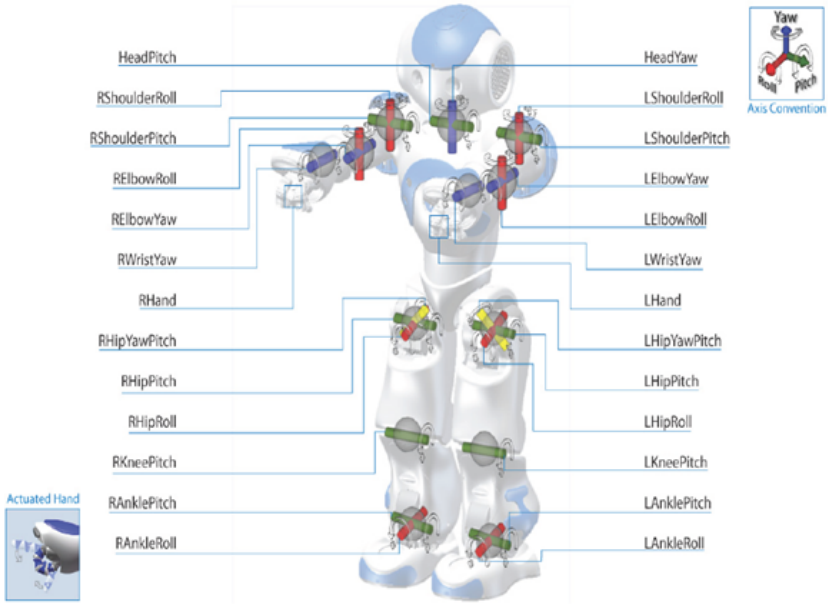

Figure 1: The joints of the robot and their axes

2 Choregraphe

Choregraphe is a standalone software which is provided with Nao by Albedaran. Choregraphe has a graphical user interface and behavior libraries by which one can control or even create Nao's movements or more

complex behaviors. It accepts Urbi and Python language; also it can directly call C++ modules developed separately. Choregraphe also has a simulated Nao in its simulator; thus, it is possible to create various kinds of movements and behaviors first using the simulated Nao, and then test them on the real robot. The motion created in Choregraphe is called a box. Choregraph's logic is based on boxes. Boxes can be connected to each other and the communication between them is performed via signals as events by using any *NaoQi* [1] modules' functions. The signal can also contains information such as a string or a number. So, it is not always possible to connect two boxes due to the kinds of their inputs and outputs.
Choregraphe can be used in two ways to program Nao:

1. Graphical Interface involving Time-line and key-frames
2. Programming codes in Python or Urbi

2.1 Graphical interface:

Choregraphe gives the possibility of creating a new motion by its graphical interface. As it can be seen in (Figure 2) each movement is composed of at least one behavior key-frame and one motion key-frame.

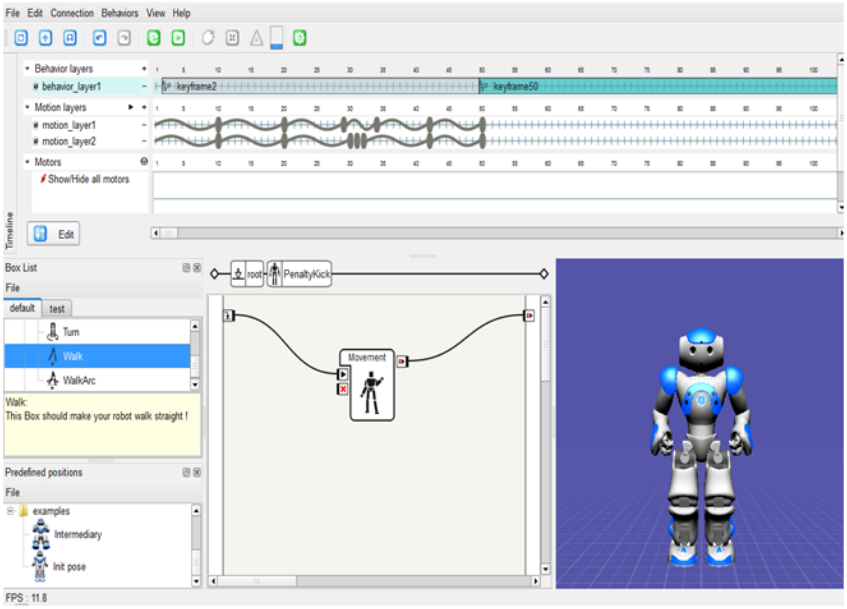

Figure 2: The graphical interface of Choregraphe: behavior and motion key-frames

On the one hand, a motion key-frame stores the values of joints of part of the body, for example, legs or hands. Moreover, motion key-frame indicates that the transition from current position to the stored position should be performed in a specific period of time. It is also possible to identify that whether this transition should be carried out linearly or by a spline function. Furthermore, Choregraphe has the ability of exporting

[1] NaoQi is based on a client-server architecture, where NaoQi itself acts as a server. The modules plug into NaoQi either as a library or as a broker, with the latter communicating over IP with NaoQi [6].

either Python or C++ code of the motion key-frames to provide the possibility of making subtle changes in the code of the movement. In addition, to figure out proper values of the joint for each key-frame, it is possible to move that part of the body of the real robot or even the simulated one into the desired position and just record the values of the joints (Figure 3).

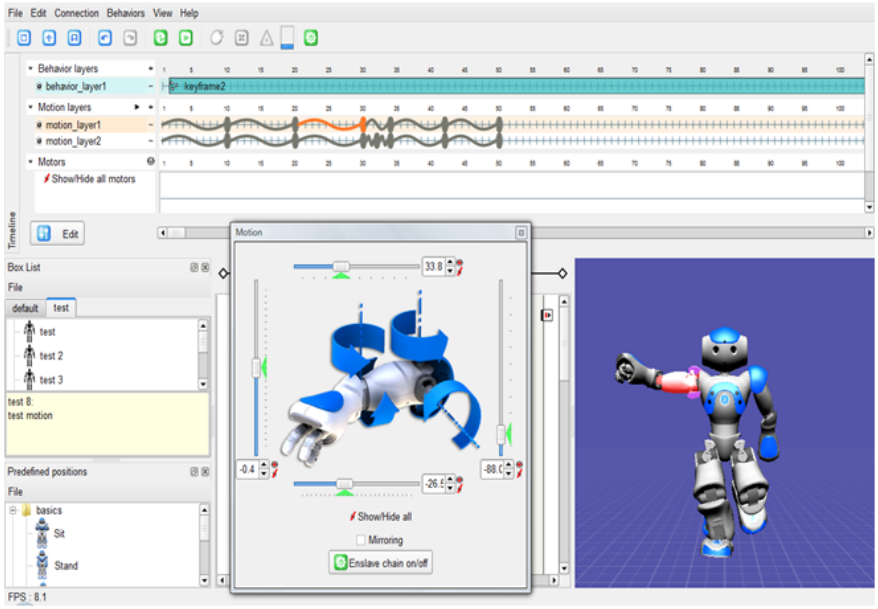

Figure 3: The graphical interface of Choregraphe: recording the motion by moving the body part

On the other hand, behavior key-frame can include either a mixture of sequences of motion's key-frames of part of the robot's body like hands, legs or head, or one or a number of created/basic motions which are connected to each other in an appropriate order (Figure 2), or even both motion key-frame and reconstructed boxes in the parallel; of course under the circumstance that they do not have conflict with each other. For instance, they should not be associated with the movement of the same part simultaneously.

2.2 Programming codes:

In Choregraphe, one can also program Nao by writing codes. Nao has its own SDK framework called NaoQi SDK which has a distributed environment containing various modules which communicate among them and perform various tasks. These modules contain inbuilts methods for various purposes. NaoQi SDK is a cross language, so one can program and control simulated Nao in Python, C++, Urbi. Methods from these modules can be executed in parallel or sequential or event driven. NaoQi SDK is a multi-platform framework and it supports:

- Linux OS
- Windows XP
- Mac OS

Using these methods from various modules, one can write a program for behavior boxes of Nao. One of the commonly used module in our project work is ALMotion. ALMotion, as the name suggests, handles motion and movement related tasks of Nao. This module has various inbuilt methods like setAngle(string pJointName, float pAngle) this sets the angle of a particular joint, getAngle(string pJointName) this gets the sensed angle of the given joint, changeBodyAngle(vector¡float¿ pAngleChanges) this method changes the angles of each joints in the body of Nao relative previous joints angle, doMove(array pJointNames, array pAngles, array pTimes, int pInterpolationType) this methods interpolates a sequence of timed motion and similarly we have more methods in ALMotion module which allow us to control the movement of joints using various motion related parameters. So these methods from different modules can be used to create behavior boxes. Below is an example of a simple code using inbuilt methods, this code makes Nao walk 1 meter. The code is written in Python. Combining these methods one can create a very complex behaviour. For example,

```
class MyClass(GeneratedClass)\:

def __init__(self):
GeneratedClass.__init__(self)
def onUnload(self):
self.onInput_onStop()

def onInput_onStart(self):
ALMotion.addWalkStraight(1, 35)
ALMotion.walk()
self.onStopped()

def onInput_onStop(self):
ALMotion.clearFootsteps()
```

Example code for walking 1 meter distance

As mentioned above that Choregraphe has a simulated Nao in its simulator. But in that simulated Nao one can not use all of the modules as in the simulation LED lights and Touch sensors of Nao is not supported so for testing these modules one has to work directly with a real Nao.

Choregraphe a feature using which one can also export codes from a box which is created using the Timeline method. And this code can be extracted in Python or C++. This feature is very useful in improving results and hence the motion or behavior which we want. Also one more important thing to notice that while using a real Nao with Choregraphe, one should make sure that both i.e. real Nao and Choregraphe have same version of NaoQi in them. The problem one may encounter if NaoQi is not same in both of them is that the method or module used in a program may not be compatible with or available in Nao.

Choregraphe has script editor in which programming is done. (Figure 4) shows the Choregraphe script editor with code of walking.

3 B-Human

In 2009, the winner of the robocup tournament was B-Human team from University of Bremen and German Research center for Artificial Intelligence [2]. Nao has a central memory where modules can share data. Every module can subscribe to the shared data and find about changes in data, which helps modules in asking Nao to perform various tasks or behaviors. B-Human is written as a separate module and not

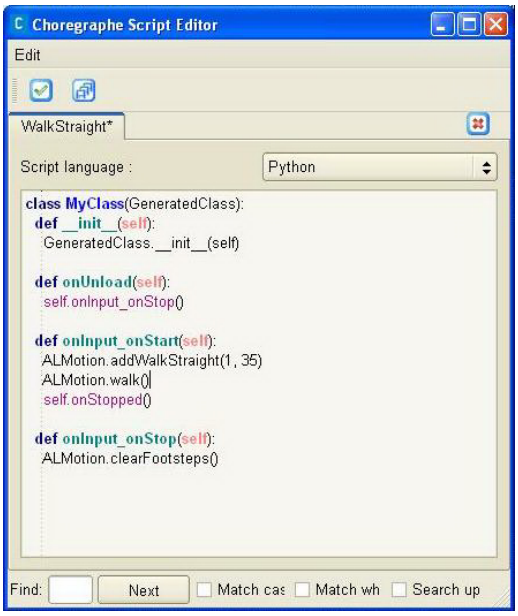

Figure 4: Choregraphe Script Editor

inside NaoQi. B-Human gets sensory data from the shared memory location. These data are written by libbhuman which a basic module implemented by B-Human team, libbhuman gets these data from DCM (Device Communication Manager) and later libbhuman access the actuators and sensors from bhuman which encapsulates the B-Human framework.

B-Human consists of various modules which performs different tasks like walking, detecting ball etc. The main focus in this project is modifying and improving the kicking technique of B-Human team. Some kinds of motions like kicking or standing up are referred as Special actions in B-Human. These special actions are implemented in a separate module called SpecialActions. All the actions in SpecialActions module are hardcoded in MOF files. For each special action there is a MOF files which contains a sequence of data elements in which each data element consists of joints data and other parameters like stiffness, time duration, data interpolation etc. For running an action Nao iterates over each data elements sequentially using the joints value in them and other parameters.

B-Human's package uses the physical robotics simulator, SimRobot. SimRobot is helpful in developing programs for Nao and testing it in a simulator before trying in real Nao. In SimRobot within a simulation one can have simulated Nao which can be controlled by giving commands and do various things. One can also connect to real Nao by using wireless or wired LAN and run commands in it and perform various task. Also by remotely connecting to a real Nao in SimRobot one can access the sensors' data and behaviors of the robot could be examined. In SimRobot we have a console using which one can give commands to the simulated Nao or the Nao to carry on the program. From the command one can ask Nao to follow a particular behaviour which is called an option, these options are modeled using *Xabls* (Extensible Agent Behavior Specification Language). The command used for executing an option or a behaviour is **xo** followed by any defined option name. For example, `xo look\_down` makes Nao look down.

4 Implementation

Investigating the statement of the problem and studying the report of the B-Human team and their team description, we attempted to read the code and install. Since the code was cross platform, we would prefer to use the Microsoft Visual studio 2008 to open, compile and run the code on Windows. Although we also tried to do this on Linux Kubuntu 9.10, because there was a conflict between one of the required package and an installed package of the Linux, we were not able to compile the code on Linux. At the beginning, we also had problem to compile the code on Windows due to existing some problems with Cygwin–which is the cross platform shell to execute Linux shell commands on Windows–which Visual Studio needed it to make and build the code.

By taking account into the compiling problems and also the time limitation, we decided to work with Choregraphe–which was much easier to be installed and to work with on Windows–to design the kicking motion first and figure out the joints values and then integrate the values obtained from Choregraphe with the B-Human code. In addition, because the facilities that Choregraphe provides, it was more tangible and has more insight to work with it to generate the movement.

To create an efficacious kicking motion, we had to figure out how many steps we needed to construct and which posture in each step. We also had to consider the kinematic chain of motions to obtain a stable and balance movement. In this manner, we create a movement with seven-step the legs' motion in parallel with six-step the arms' motion primarily. But, to integrate the result with the B-Human's code, we had to split some steps to two to have eight steps for both the legs and arms the same as the code; the sequence of the motions is as bellow:

1. stand
2. lean to the side
3. lift leg
4. leg behind
5. leg front
6. knee open
7. leg beside
8. stand

where stand step is the same as standing position during walking; just legs are opened more to gain a more balance state. Thus, it takes lesser time for transition from the previous movement, which is most of the time walking behind the ball, to the kick action, or starting walking after kicking the ball. We also decrease the duration time and set the transition method, from lifting the leg to opening the knee, linear to make the robot move faster. This helps the robot to kick also stronger. In addition, in our kicking motion, the robot lifts its leg to a height to hit a point of the ball around its center which is effective to make the ball go farther.

Moreover, in order to keep the balance, the robot uses its arms' movements beside leaning a bit more to the side. By moving arms to front/back or open/close, the robot was able to be in more stable/balance state.

After designing the kick in Choregraphe, we have to implement it in B-Human. But Choregraphe and B-Human are independent from each other as both of these are stand alone softwares. As the kick is hard

coded in B-Human we need the values of kick which consists of joints data in a sequence. The joints data is extracted from the code written in Choregraphe and transformed in a format usable for B-Human.

The format in which both Choregraphe and B-Human reads the joints data as inputs have differences. In choregraphe itself one can see differences in the format of joints data used depending on which method is used for programming. Like while using time-line or graphical interface the joints data are in degrees. And if we use the programming interface of Choregraphe which is programmed in either Python or Urbi, the joints data are required to be in radians. Now for B-Human SimRobot, where we test our kick first, joints data are required in degrees. Also in the experimentation, it was found that besides the difference in the format of joints data there were other differences also. The order of joints data as input in Choregraphe and in B-Human SimRobot is different and also the axes of joints data has to be transformed if one wants to use a data set from one system to other system.

Order of joints data in choregraphe is following:

["HeadYaw", "HeadPitch", "LShoulderPitch", "LShoulderRoll", "LElbowYaw", "LElbowRoll", "LWristYaw", "LHand", "LHipYawPitch", "LHipRoll", "LHipPitch", "LKneePitch", "LAnklePitch", "LAnkleRoll", "RHipYawPitch", "RHipRoll", "RHipPitch", "RKneePitch", "RAnklePitch", "RAnkleRoll", "RShoulderPitch", "RShoulderRoll", "RElbowYaw", "RElbowRoll", "RWristYaw", "RHand"]

Following is the order of joints data in B-Human SimRobot with Choregraphe's joint name in bracket:

["headPan (HeadYaw)", "headTilt (HeadPitch)", "armLeft0 (LShoulderPitch)", "armLeft1 (LShoulderRoll)", "armLeft2 (LElbowYaw)", "armLeft3 (LElbowRoll)", "armRight0 (RShoulderPitch)", "armRight1 (RShoulderRoll)", "armLeft2 (RElbowYaw)", "armRigmt3 (RElbowRoll)", "legLeft0 (LHipYawPitch)", "legLeft1 (LHipRoll)", "legleft2 (LHipPitch)", "legLeft3 (LKneePitch)", "legLeft4 (LAnklePitch)", "legleft5 (LAnkleRoll)", "legRight0 (RHipYawPitch)", "legRight1 (RHipRoll)", "legRight2 (RHipPitch)", "legRight3 (RKneePitch)", "legRight4 (RAnklePitch)", "legRight5 (RAnkleRoll)"]

As we are using data from Choregraphe in B-Human, so we need the transformations of axes from Choregraphe to B-Human. The transformations needed are following:

```
LShoulderPitch = 90 - LShoulderPitch x DEG
LShoulderRoll = LShoulderRoll x DEG - 90
LElbowYaw = LElbowYaw x DEG
LElbowRoll = LElbowRoll x DEG
LHipYawPitch = LHipYawPitch x DEG
LHipRoll = 0 - LHipRoll x DEG
LHipPitch = LHipPitch x DEG
LKneePitch = LKneePitch x DEG
LAnklePitch = LAnklePitch x DEG
LAnkleRoll = LAnkleRoll x DEG
RHipRoll = RHipRoll x DEG
RHipPitch = RHipPitch x DEG
RKneePitch = RKneePitch x DEG
RAnklePitch = RAnklePitch x DEG
RAnkleRoll = RAnkleRoll x DEG
RShoulderPitch = 90 - RShoulderPitch x DEG
RShoulderRoll = 0 - 90 - RShoulderRoll x DEG
```

```
RElbowYaw = RElbowYaw x DEG
RElbowRoll = 0 - RElbowRoll x DEG
```

where **DEG** = $\frac{180.0}{\pi}$ a factor for changing choregraphe's radian joints values into degrees.

5 Conclusion

Succinctly, there are a lot of tools by which one can work with the Nao robot and program it. In this case, Choregraphe is very easy to work with and gives efficient facilities to control the robot's motions and behaviors, such as a graphical interface or a coding editor. By using Choregraphe, we have obtained a set of effective values of the robot's joints which generates a kicking movement that, practically, seems to be more stable than the original kick in the B-Human code due to handling the balance by moving the arms (which is not utilized in the B-Human's kick design) and relative legs' position. Also the kick is made faster by setting the time of performing the motions' steps shorter along with a linear transition at the point of time the ball is being touched. This fastness not only helps the robot to act better in real time matches, but it also entails to having a stronger kicking and longer distance shoot.

In addition, the script, we used to map the values from Choregraphe to the B-human code, can be used more to design other actions also in Choregraphe and put them in the code then.

For the future work, we would like to improve our kicking motion by evaluating the effective values of the joint stiffness; it seems that by changing the hardness of some joints during kicking, a stronger kick can be obtain. Furthermore, robot will perform better if it is able to kick the ball in a specific direction; this goal can be achieve by applying Machine learning methods like *Reinforcement Learning* to learn the robot how much it should open its legs related to the ball's position to be able to direct the ball to the specified point.

References

[1] http://www.b-human.de/download.php?file=coderelease09_doc

[2] http://www.b-human.de

[3] http://www.robocup.org

[4] http://www.aldebaran-robotics.com/en

[5] http://www.tzi.de/spl/bin/view/Website/WebHome

[6] http://cgi.cse.unsw.edu.au/~cs4411/wiki/index.php?title=Aldebaran_Nao