Soccer Skills for Humanoid Robots

Georgios F. Pierris

Department of Electronic and Computer Engineering

Technical University of Crete, Greece

Thesis Committee

Assistant Professor Michail G. Lagoudakis

Assistant Professor Aikaterini Mania

Assistant Professor Nikolaos Vlassis

ΠΟΛΥΤΕΧΝΕΙΟ ΚΡΗΤΗΣ

Δεξιότητες Ποδοσφαίρου για Ανθρωποειδή Ρομπότ

Γεώργιος Φ. Πιερρής

Τμήμα Ηλεκτρονικών Μηχανικών και Μηχανικών Υπολογιστών

Χανιά, Απρίλιος 2009

This thesis is dedicated to my parents, who were always on my side,
accepting my choices, congratulating my successes, but also
supporting my faults ...

Acknowledgements

Since the very first time I started reading scientific documents, I was against the idea of acknowledging the people you love and who unselfishly support you by just dropping a few lines at them. Time has gone by and right now after having completed my thesis, I am definitely sure that I have not been completely fair to them.

I would like to thank my friends Nikiforos, Vasilis, Ilias, and Fotis for their support and the great moments we shared in Chania. Everything would be different, if not for them and I am pretty sure we'll manage to continue having some great time in the following years.

Furthermore, I would like to thank my teammates from the team Kouretes; the former members Petros, Giorgos, Suzanna, Chrisaygi who helped us in our very first steps in the team, but especially the current members Andreas, Alex, Lefteris, and last but not least Vagelis, whose speculating behavior, his judging eye, and his supportive attitude towards my work made my life a living nightmare in a productive way making the outcome of this work great indeed. The aforesaid "guys" are far beyond teammates and I consider them as friends. That was one of the reasons I always enjoyed being a member of Kouretes Team.

Words are simply not enough for Prof. Michail Lagoudakis. I still remember the first Thursday, when we met at the Intelligence Laboratory, introducing me to Kouretes Team, spending that afternoon on how to use the memory stick of the Aibo robot. That was the beginning of an exciting journey. We' ve lived so much together and I've earned a lot more from this collaboration. His help and support was abundant in all aspects of my work and personal development.

I would also like to thank Prof. Nikos Vlassis, whose support in our team, solutions, theoretical background, and his extensive work on the Simulation League with Stathis and Daisy has improved our work as well.

The leaps I have made in the last 3 years would have been only small steps, if it wasn't for Helen. She was and still is so important in my life, that I really can't imagine how it would be without her. Thank you.

I would lastly like to thank my two sisters, Theoni and Anastasia, who were always by my side teaching me how life works; concepts that sculptured my personality throughout years. Fotis, Vasilis and Elena are the three gifts they gave me. Being an uncle has changed my way of thinking and I'll do my best in the future to ensure you have anything you need. I hope you one day spend some time to read this document.

Thank you all ...

Abstract

Low-cost robots with large number of degrees of freedom are becoming increasingly popular, nevertheless their programming still remains a domain for experts. This thesis introduces the *Kouretes Motion Editor* (KME), an interactive software tool for designing complex motion patterns on robots with many degrees of freedom using intuitive means. KME allows for a TCP/IP connection to a real or simulated robot, over which various robot poses can be communicated to or from the robot and manipulated locally using the KME graphical user interface. This portability and flexibility enables the user to work under different modes, with different robots, using different host machines. KME was originally designed for and currently supports only the Aldebaran Nao humanoid robot which features a total of 21 degrees of freedom. This work helped us develop various much needed soccer skills for the Nao robot. KME has been employed successfully by *Kouretes*, the RoboCup team of the Technical University of Crete, for designing various special actions, thanks to which the team ranked in the 3rd place at the RoboCup 2008 competition (Standard Platform League).

Περίληψη

Τα προγραμματιζόμενα ρομποτικά συστήματα χαμηλού κόστους, γίνονται ολοένα και περισσότερο δημοφιλή και προσιτά στους κοινούς χρήστες, ωστόσο ο προγραμματισμός τους παραμένει ακόμη ένας τομέας για ειδικούς. Η παρούσα διπλωματική εργασία περιγράφει το σχεδιασμό, την ανάπτυξη και τη χρήση του Kouretes Motion Editor (KME), ενός προγραμματιστικού εργαλείου που επιτρέπει το σχεδιασμό πολύπλοκων μοτίβων κίνησης σε ρομποτικά συστήματα με πολλούς βαθμούς ελευθερίας, χρησιμοποιώντας διαισθητικές μεθόδους. Το KME επιτρέπει τη δημιουργία δικτυακών (TCP/IP) συνδέσεων με ένα πραγματικό ή προσομοιωμένο ρομπότ, διαμέσου των οποίων μπορούν να διακινηθούν εύκολα διάφορες διατάξεις των ρομποτικών αρθρώσεων από και προς το ρομπότ, να επεξεργαστούν τοπικά μέσα από το γραφικό περιβάλλον του KME , και να σχεδιαστούν σύνθετες κινήσεις ως χρονισμένες ακολουθίες τέτοιων διατάξεων. Η μεταφερσιμότητα και ευελιξία που παρέχεται από το εργαλείο, επιτρέπει στον χρήστη να εργάζεται με διαφορετικούς τρόπους, με διαφορετικά ρομπότ, σε διαφορετικές υπολογιστικές πλατφόρμες. Το KME υποστηρίζει πλήρως τη ρομποτική πλατφόρμα Nao της Aldebaran Robotics, ένα ανθρωποειδές ρομπότ με 21 βαθμούς ελευθερίας, ωστόσο μπορεί να παραμετροποιηθεί εύκολα για χρήση και με άλλα ρομποτικά συστήματα. Το KME χρησιμοποιήθηκε με επιτυχία από την ομάδα ρομποτικού ποδοσφαίρου του Πολυτεχνείου Κρήτης *Κουρήτες* για την ανάπτυξη διαφόρων ποδοσφαιρικών δεξιοτήτων για το ρομπότ Nao στα πλαίσια του διεθνούς διαγωνισμού ρομποτικού ποδοσφαίρου RoboCup. Η κατάκτηση της 3ης θέσης στο πρωτάθλημα Standard Platform League στο RoboCup 2008 που διεξήχθη στο Suzhou της Κίνας από την ομάδα *Κουρήτες* οφείλεται σε ένα μεγάλο βαθμό στις άρτιες δεξιότητες των ρομποτικών παικτών της.

Contents

List of Figures

Chapter 1

Introduction

Robots can get people confused on what exactly someone means. Robotics is such a general, diverse field including a lot of disciplines, that elaborating on every available detail on the exact problem we want to solve can be considered obligatory.

1.1 Approaching The Problem

In this thesis, robots are considered as electromechanical machines having actuators giving them the ability to act, operate in, or even alter the environment they live in.

Nowadays more than ever, this type of robots are becoming increasingly popular. Detecting the reason is rather trivial and can be found in technology advancements, which force production cost to follow a steep downslope. Keeping cost in low levels universities, companies, industries and even individuals consider more easily buying robots.

In the last category, low-cost robots can be bought for a few thousands of Euros, but their abilities cannot be considered limited by any means. Some examples include the Sony Aibo robot featuring 20 degrees of freedom, the Aldebaran Nao (1) with 22 degrees of freedom, the Bioloid Kit which can be assembled in any formation the user wants, and many other examples. Nevertheless, programming these robots still remains a domain for scientists, researchers, and experts who have previous experience at least in the basic concepts of robotics.

In order to avoid all this complexity, we have developed *Kouretes Motion Editor* (from now on KME), an interactive tool for designing complex motion patterns on robots with many degrees of freedom. The initiative of this work, can be found back in motion designing in the Aibo robot, which was a very tedious task. KME 's philosophy can be considered straightforward, allowing user have multiplatform libraries, giving them the ability to have a multiplatform tool working under Windows, Linux and MacOS.

KME allows for a TCP/IP connection to a real or a simulated robot (Webots for now and Microsoft Robotics Studio in the near future), over which various robot poses can be communicated to or from the robot and manipulated locally using the KME graphical user interface. Using generic modules, we are able to use any operating system, communicating with any robotic platform, writing some minor code in the desired language. This portability and flexibility enables the user to work under different modes, with different robots, using different host machines. KME was originally designed for and currently supports only the Aldebaran Nao humanoid robot, but some work has been done to port it for the Bioloid Kit.

This work helped us develop, various much needed soccer skills for the Nao robot. These motion patterns include a standing up routine, a goalkeeper behavior, a kick motion and other special actions. KME has been employed successfully by *Kouretes*, the RoboCup team of the Technical University of Crete, in the design of the aforesaid motions. *Kouretes* team ranked in the 3rd place at the RoboCup 2008 competition (Standard Platform League), and a part of this achievement can be accredited to KME .

1.2 Thesis Outline

Chapter 2 provides some background information on the RoboCup competition and especially on the Standard Platform League. This chapter is mainly intended to the reader, who does not have any previous experience or knowledge of RoboCup. It is a rather introductory chapter, presenting the basic ideas, subsequently should you need any further details, please refer to the bibliography.

In Chapter 3 we introduce the newly developed platform, the Aldebaran Nao robot, which is the descendant of the Sony Aibo robot, in the Standard Platform League. With the new robot coming into play a lot of things had to be redesigned, especially the motions due to the different platform.

Chapter 4 demonstrates some of the thoughts we had while developing KME ; where did we receive our inspiration, what options did we have in motion designing, how motion is perceived by robots, and other issues. All these questions are going to be answered in this chapter.

Continuing to Chapter 5 the main core of this thesis is presented, which is the development of Kouretes Motion Editor. A tool designed from scratch, in order to speed up the procedure of creating complex motion patterns in robots with large DOFs. An overall description is provided, and also the technical details, its usage and information on how to design and execute a motion already developed.

Moving on to Chapter 6, a discussion on the results is taking place, providing a rather empirical evaluation of our work, since it was tested, judged and compared against other team's work in this competitive domain.

In Chapter 7 the reader will find some of the related work in the Nao robot, or other robotic systems. We then discuss a comparison between Kouretes Motion Editor and other "competitors" tending to share the pie of robot motion design in the future.

The future work and proposals on the improvement of Kouretes Motion Editor is being discussed on Chapter 8. Things that have not been done yet, or short- and long-term ideas that come up through KME 's development, and usage can be found here.

Appendix A is intended to the users of KME . This is a technical manual for the first time user of KME , things you should know before starting, some specificities that need to be cleared and a thorough discussion on each button separately.

Appendix B is intended to be read only by those, who are going to seriously work with KME . This is a demonstration of all the details should you be aware of before going to develop your own server. A server has to be created to serve as an intermediate communication mean between KME and a robot other than Aldebaran Nao. For the latter we already provide you one.

Lastly, Appendix C gives you an overview of the format of the configuration XML file describing the new robot you want to work with. It is not obligatory to have read Appendix B before continuing to this chapter, but it is suggested to have the server developed before moving on. Building the XML file is trivial compared to developing the server which is not that hard as well, for the experienced programmer.

Chapter 2

Background

2.1 RoboCup Competition

The RoboCup Competition, in its short history, has grown to a well-established annual event bringing together the best robotics researchers from all over the world. The initial conception by Hiroaki Kitano (2) in 1993 led to the formation of the RoboCup Federation with a bold vision: "By the year 2050, to develop a team of fully autonomous humanoid robots that can win against the human world soccer champions".The uniqueness of RoboCup stems from the real-world challenge it poses, whereby the core problems of robotics (perception, cognition, action, coordination) must be addressed simultaneously under real-time constraints. The proposed solutions are tested on a common benchmark environment through soccer games in various leagues,with the goal of promoting the best approaches, and ultimately advancing the state-of-the-art in the area.

2.2 RoboCup Leagues

Beyond soccer, RoboCup now includes also competitions in search-and-rescue missions (RoboRescue), home-keeping tasks (RoboCup@Home), robotic performances (RoboDance), and simplified soccer leagues for K-12 students (RoboCup Junior). Broadening the research areas where RoboCup focuses, was a very interesting and clever addition, which enables all the more scientists and researchers

combine their expertise in order to solve real world problems. A lot of progress has been made so far in many disciplines of robotics and RoboCup has been established in one of the most important events around the world.

2.3 RoboRescue

RoboRescue initiated from the need of people to create robots capable of operating in hostile or inaccessible environments by humans. This area is very motivating in terms of helping the humanity while promoting robotics. Multi-agent team work coordination, physical robotic agents and rescue strategies are all being tested in real or simulated environments (Figure 2.1), using sensors including cameras, temperature sensors, CO_2 sensors and other that allow for fast and efficient human body localization.

Figure 2.1: RoboRescue environments can become extremely hostile for robots.

2.4 RoboCup@Home

The RoboCup@Home league aims to develop service and assistive robot technology with high relevance for future personal domestic applications. It is the largest international annual competition for autonomous service robots and is part of the

RoboCup initiative. A set of benchmark tests is used to evaluate the robots' abilities and performance in a realistic non-standardized home environment setting (Figure 2.2).

Most of the research lies in many domains including Human-Robot-Interaction and Cooperation, Navigation and Mapping in dynamic environments, Computer Vision and Object Recognition under natural light conditions, Object Manipulation, Adaptive Behaviors, Behavior Integration, Ambient Intelligence, Standardization and System Integration.

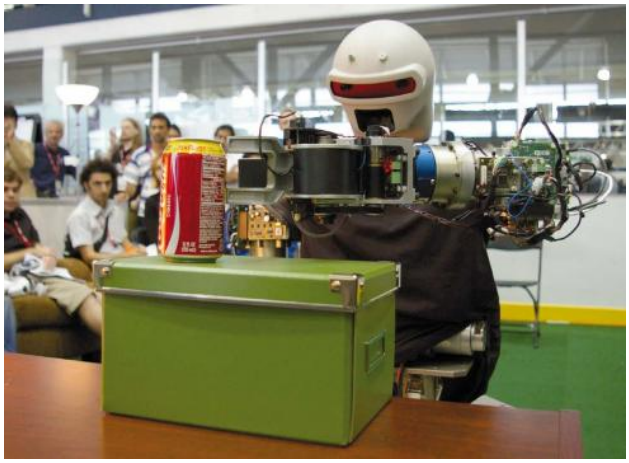

Figure 2.2: Robocup@Home represents a social aspect of robotics interacting with people.

2.5 RoboCup Junior

Robocup Junior is one of the smaller and less competitive, in terms of antagonism sub-domain of RoboCup, which is mainly intended in the preparation of gifted children interested in robotics. It is very important to understand that this competition has nothing to be jealous of the other leagues, but in fact share the same vision and the required dedication to excel.

2.5.1 Dance

RoboDance might one say to be the most amusing competition of RoboCup, where robots are performing in front of their audience. Dances, most of times, are innovative and unique. Developers are being judged on the novelty of motions, the cooperation of the robots, and finally the synchronization of the robot motion according to the music.

2.5.2 Rescue

Rescue in the Junior league, is a simplified version of RoboRescue and is limited in the line-following problem. Each team has to qualify from a number of tracks whose difficulty gradually increases. The trials are judged on the duration of the track completion and on the agent's behavior in misleading parts of the track, such as line intersections, or line gaps.

2.5.3 Soccer

The soccer competition is played by two cylindrical robots which share the same features and play soccer in a box or in a table surrounded by low walls. This is by far the most exciting competition in Robocup Junior and the most demanding.

2.6 RoboCup Soccer League

The RoboCup Soccer League, is the domain with the most fans. In this league researchers combine their technical knowledge in order to prepare the best robotic soccer team among other universities.

2.6.1 The Standard Platform League

Standard Platform League (SPL) of the RoboCup competition is the most popular league, featuring two to four humanoid Aldebaran Nao robot players in each team. This league was formerly known as the Four-Legged League with Sony Aibo robots, which were replaced in 2008 by Aldebaran Nao (Figure 2.3). Games take place in a $4m \times 6m$ field marked with thick white lines on a green carpet. The

two colored goals (sky-blue and yellow) also serve as landmarks for localizing the robots in the field. Each game consists of two 10-minute halves and teams switch colors and sides at halftime. There are several rules enforced by human referees during the game. For example, a player is punished with a 30-seconds removal from the field if he performs an illegal action, such as pushing an opponent for more than three seconds, grabbing the ball between his legs for more than three seconds, or entering his own goal area as a defender.

Figure 2.3: Standard Platform League game in Robocup 2008(Opponents should be in different colors, but there was a lack of Nao robots in that event due to malfunctions)

The main characteristic of the Standard Platform League is that no hardware changes are allowed; all teams use the exact same robotic platform and differ only in terms of their software. This convention results to the league's enrichment of a unique set of features: autonomous player operation, vision-based perception, legged locomotion and action. Given that the underlying robotic hardware is common for all competing teams, research effort has been focused on the development of more efficient algorithms and techniques for visual perception, active localization, omni-directional motion, skill learning, and coordination strategies.

During the course of the years, one could easily notice a clear progress in all research directions.

2.6.2 Simulation League

Every year there is a number of simulation games taking place in RoboCup competitions. These include 2D soccer games, where teams consist of 11 agents providing to developers with a perfect multi-agent environment to tune and benchmark their solutions. 3D simulation games exist as well; usage of physics engines demand more realistic approaches.

Simulators offer the ability to control the amount of "negative" realism added in these environments; thus, it is a great way to allow researchers work focusing in multi-agent cooperation approaches and other state-of-the-art algorithms, abstracting from real-world problems (gravity, forces etc).

In RoboCup 2008, three of the most important simulation leagues, were the official RoboCup Simulation League run in an open source simulator, the Microsoft Robotics Studio competition, and the Webots RoboStadium.

2.6.3 Small Size League

A Small Size robot soccer game takes place between two teams of five robots each. Each robot must fit within an 180mm diameter circle and must be no higher than 15cm, unless they use on-board vision. Robots play soccer on a 6.05m long by 4.05m wide, green carpeted field with an orange golf ball. Vision information is either processed on-board the robot or is transmitted back to the off-field PC. Another off-field PC is being used to communicate referee commands and position information to the robots, when an extra camera mounted on top of the field serves as the vision sensor. Typically, off-field PCs are used in the coordination and control of the robots. Communication is wireless and typically use dedicated commercial FM transmitter/receiver units[1].

[1] http://small-size.informatik.uni-bremen.de/rules:main

2.6.4 Middle Size League

Middle Size is more competitive and demanding, having the largest field dimensions among other leagues (Figure 2.4). Two teams of mid-sized robots consist of 5 players each, with all sensors on-board play soccer on a field of $18m \times 12m$, whereas relevant objects are distinguished by colors only. Communication among robots (if any) is supported on wireless communications. Once again, no external intervention by humans is allowed, except to insert or remove robots in/from the field. These robots are the best players far among other leagues. The robots' bodies are heavy enough having powerful motors, heavy batteries, omni-directional camera, and a full laptop computer running in every robot; characteristics that make this league a great domain for research.

Figure 2.4: Middle size league game in Robocup 2008.

2.6.5 Humanoid League

The Humanoid League is one of the most dynamically progressing leagues and the one closest to the 2050's goal. In this league, autonomous robots with a human-like body plan and human-like senses play soccer against each other. In addition

to soccer competitions, technical challenges take place. The robots are divided into two size classes: KidSize (30-60cm height) and TeenSize (100-160cm height). Dynamic walking, running, and kicking the ball while maintaining balance, visual perception of the ball, other players, and the field, self-localization, and team play are among the many research issues investigated in this league. Several of the best autonomous humanoid robots in the world compete in the RoboCup Humanoid League[1].

2.7 Kouretes Team

Kouretes Team, is a collaboration of the Intelligent Systems Laboratory at the Department of Electronic and Computer Engineering, and the Intelligent Systems and Robotics Laboratory at the Department of Production Engineering and Management. It was the first Greek team participating in Robocup competitions, specializing in the Standard Platform League and the MSRS Simulation League. The team's name, *Kouretes*, comes from the Ancient Greek Mythology. Formerly consisted by four AIBO robots named Epimedes, Paionaios, Iasios, and Idas, after the five Kouretes brothers, who were ancient Cretan warriors.The fifth brother, Hercules, represents all the human members of the team. Since Spring 2008 four NAO robots joined the team, but they have not been named yet.

Kouretes have participated in many competitions, exhibitions, and affairs, but the highlights of the steep orbit the team followed were the second place in Robocup 2007, Atlanta, USA in the MSRS Simulation League and the first and third place in Robocup 2008, Suzhou, China in the MSRS Simulation League, and the Standard Platform League accordingly.

Kouretes team's 2008 formation in Figure 2.5 from left to right in the front row are Andreas Panakos (SPL), Daisy Chroni (Simulation Team), Alexandros Paraschos (SPL), Stathis Vafias (Simulation Team), and in the back row Professor Michail G. Lagoudakis (Kouretes Team Leader) and Georgios F. Pierris (SPL). Lastly but not least, we must mention the other two members of *Kouretes*

[1] http://www.tzi.de/humanoid/bin/view/Website/WebHome

supporting the team from our base, Lefteris Chatzilaris(Simulation Team Robostadium), and Evangelos Vazaios(SPL), who entered the team back in 2008, working now actively and full time, looking forward to Robocup 2009 competition in Graz, Austria.

Figure 2.5: Kouretes team 2008 formation. From left to right in the front row are Andreas Panakos(SPL), Daisy Chroni(Simulation Team), Alexandros Paraschos(SPL), Stathis Vafias(Simulation Team), and in the back row Professor Michail G. Lagoudakis(Kouretes Team Leader) and Georgios F. Pierris(SPL).

More information and news of the team but also its members can be found at http://www.kouretes.gr.

Chapter 3

NAO : The New Platform

Discontinuation of the Sony Aibo robot in 2006, and the large number of problems created by Sony's decision, forced the Robocup Federation to conduct an open call to robotics companies. They were asked to present their offers for a new robotic platform for the Standard Platform League. Right after the end of Robocup 2007, in Atlanta, USA the votes were for a fairly new company, Aldebaran Robotics based in Paris, which was developing a revolutionary 21 DOF humanoid robot.

3.1 NAO's Description

Nao, in its final version, is a 58 cm, 4.3 Kg humanoid robot. To this time, Nao has not been released commercially yet, however Aldebaran's goal is to eventually promote Nao as an educational robotic platform and a family entertainment robot affordable to most budgets. The initial limited edition of the robot (RoboCup edition v2) made its debut at RoboCup 2008. The Nao robot carries a full computer on board with an x86 AMD Geode processor at 500 MHz, 256 MB SDRAM, and 1 GB flash memory running an Embedded Linux distribution. It is powered by a 6-cell Lithium-Ion battery which provides about 45 minutes of continuous operation and communicates with remote computers via an IEEE 802.11g wireless or a wired ethernet link. The Nao robot features a variety of sensors and actuators.

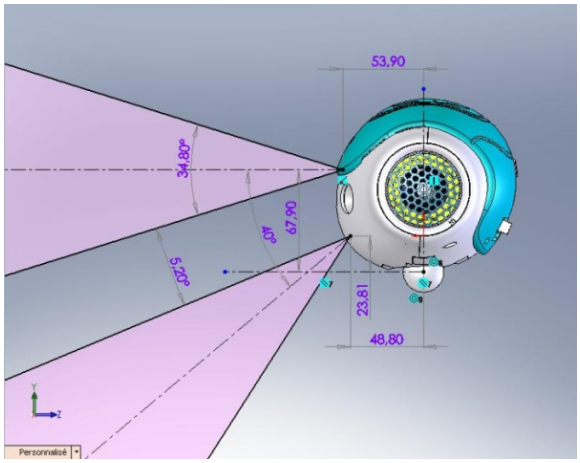

Figure 3.1: Nao's field of View

3.1.1 Cameras

In the NAO V2 robot, only one camera was mounted in the head. It was a 30fps, 640×480 color camera, provided with a rich API, allowing us many interventions, in order to get the best result, under any reasonable lighting conditions. Unfortunately, probably a driver bug caused us a lot of problems, since the camera output was in bytes in YUV422 format. The ordering was of U, Y1, V, Y2, however sometimes a shift in this order flipped the U and V in the resulting image. Even though, most colors were unaffected, orange was light purple in the output. Some solutions to this problem have been suggested by GT/CMU, rUNSWift and other, but this problem finally solved in the V3 robot by Aldebaran.

During the first games, it was clear that NAO did not have a clear view in the short frontal area. Even when the Head Pitch Joint angle was at its lowest value, the ball was not visible and its position was not determinable according to the NAO's legs. Thus, most teams were obliged to create a bend motion, not only consuming gameplay time, but also stressing the NAO's joints. The answer from Aldebaran, was to mount a second identical camera in the head, pointing directly in the short frontal area. In the NAO V3, these cameras are mounted on the head in vertical alignment providing non-overlapping views of the lower and distant frontal areas (Figure 3.1), but only one is active each time and the view

can be switched from one to the other almost instantaneously.

3.1.2 Connectivity

Connecting to a robot is a major concept, thus Aldebaran provide us three means of connection. The NAO robot, can be connected directly or sharing the same network, through a wired ethernet link. Additionally, an IEEE 802.11g wireless card is available, which is the frequent way of connection, due to the lack of forces applied to the robot through the network cable, and finally Aldebaran provides a serial cable, which comes into play when we want to debug the robot, and read messsages concerning the Nao's procedures.

3.1.3 Audio, Sensors, Various

A pair of microphones allows for stereo audio perception. Two ultrasound sensors on the chest allow Nao to sense obstacles in front of it and a rich inertial unit (a 2-axis gyroscope and a 3-axis accelerometer) in the torso provides real-time information about its instantaneous body movements. Finally, an array of force sensitive resistors on each foot delivers feedback on the forces aplied to the feet, while encoders on all servos record the actual joint position at each time and two bumpers on the feet provide information on collisions of the feet with obstacles. The Nao robot has a total of 21 degrees of freedom; 4 in each arm, 5 in each leg, 2 in the head, and 1 in the pelvis (there are 2 pelvis joints which are coupled together on one servo and cannot move independently). Stereo loudspeakers and a series of LEDs complement its motion capabilities with auditory and visual actions.

3.1.4 NaoQi

The Nao programming environment is based on the proprietary NaoQi framework which serves as a middleware between the robot and high-level languages, such as C, C++, and Python. NaoQi offers a distributed programming and debugging environment which can run embedded on the robot or remotely on a computer offering an abstraction for event-based, parallel and sequential execution. Its

architecture is based on modules and brokers which can be executed onboard the robot or remotely and allows the seamless integration of various heterogeneous components, including proprietary and custom-made functionality.

3.2 NAO Hardware

The Nao has 21 revolute joints all of which are encoded motors. Particularly, the robot can be split in five major joint chains which are :

Head

The head has two degrees of freedom; pitch and yaw serving the camera orientation needs.

Left Arm

Each arm chain, contains 4 degrees of freedom including shoulder's pitch and roll, and elbow's yaw and roll. In the Academics version there are plans of adding the wrist yaw and enabling the fingers of the robot, which are currently not functional in the Robocup edition.

Right Arm

Same joints as in the left arm.

Left Leg

The degrees of freedom in each leg are 6, summarizing them to hip's yaw-pitch(two DOFs serverd by one joint), hip pitch and roll, knee's pitch, and ankle's pitch and roll.

Right Leg

Same joints as in the left leg.

If we sum up all the above joints, we get a number of 22, which is virtual, due to a mechanical limitation; Left Hip Yaw Pitch and Right Hip Yaw Pitch are physically connected and cannot be controlled independently. Thus, we have a total sum of 21 degrees of freedom.

3.3 Conclusion

In overall, Aldebaran Robotics designed and assembled a low-cost robot, which can be a great, not only scientific, but also entertaiment platform, easily programmable, focusing on the wide audience of robotics' researchers and fans. The development of humanoid robots is a tough procedure that only few universities and companies have undergone, and even fewer were located in Europe.

To be fair, we have to admit that the first version of Nao was not functional to the level Robocup teams would like it to be. Nevertheless, most teams were able to present some basic soccer behaviors, confirming that even under these limitations and the minimum available time for development, people involved in Robocup gave their best shot.

Nowadays, in February 2009, code development in Nao V3, has become a less tedious work, and we are all looking forward to take the best out of this platform, in order to make our contribution in the RoboCup community.

Chapter 4

Soccer Skills For Humanoid Robots

4.1 Imitation

When it comes to robotics, researchers, used to and still do, receive most of their inspiration by nature. Taking advantage of nature's evolution, we can assume that by copying more or less this behavior or organisms, we indirectly use a shortcut in robotics evolution. Additionally, the creation and development of a bio-inspired robot is a straightforward approach,in terms of design, and furthermore ...it is funnier.

In that way, choosing a humanoid robot to be used in the Standard Platform League was not a choice "coming out of the hat". Right now, we are sure that heavy wheeled robots such as the ones used in Middle Size League are performing a lot better, but using humanoid robots is a more difficult, competitive, and demanding task that yields researcher's need to work under this field.

4.2 Human Soccer

Soccer played by humans, is our first option to search for solutions adaptable in our robotic players. Since humanoids robot share pretty much the same degrees of freedom as people have, trying to figure out totally new motions, or invent an

arbitrary pattern from scratch, are just thoughts popping out of our head after some unsuccessful trials of human motions imitations (Figure 4.1).

Figure 4.1: To play soccer, some extra skills are required; these are absent not only in robots, but in a vast number of humans as well [picture taken from www.wldcup.com]

Soccer is actually, a part of everyones life, and even when someone dislikes it, probably everybody has played at least once in his lifetime, or spent an afternoon watching a game. Obviously, research done to understand which motions are much needed was straightforward. Actions taking place during a game include walking, kicking, goalkeeper diving, standing up, and others. Unfortunately, for a programmer to create all these motions is time consuming, and tiring.

4.3 Large DOF's Robots

Robots can be by definition not only physical robots, but also virtual software agents. We, in this thesis, and in the bounds Kouretes Standard Platform team is mainly interested in, are working on real or simulated robots.

4.3.1 Problems In Motion Design

Working with robots featuring large number Degrees Of Freedom result in a complex space when it comes to motion patterns. For example, Nao robot has 21 DOFs available, thus in order to create a configuration of the robot, programmers have to set a value for each one of the available joints, perhaps in a vector, plus one variable for the time to execute the transition from one configuration to the other. Now, we have to create one of the previously mentioned vectors for each configuration we want.

Summarizing, creation of a simple motion formed by three poses, would demand $(21\ joints + 1\ time) \times\ 3 = 88\ values$. Using common sense, we understand that choosing these values by hand is not plausible in many cases and even if we were able to do this, tuning a pose already developed is not a good option, especially considering the time users should have spend to fullfill this task.

Another problem faced by robotics researchers involved in motion designing is the adaptability of the robot's motion patterns in dynamic environments, not only by means of adapting in various terrains or external forces applied to it, but also adapting in minor alternations of the environment. Minor changes such as using different types of carpets, are problems that may be surpassed by tuning each motion pattern every time. Tuning can be usefull in some circumstances, but not having an automated way of doing it, can lead to lots of waisted hours simply reinventing the wheel.

Furthermore, no matter how strong our desire is to create identical robots, till now the existence of mechanical differences among the very same robots, is present even in construction line. Furthermore, motors, plastic parts, circuits etc. undergo some gradually increasing tolerance, since robots are subject to many and demanding tests in research environments. These variations might lead to unexpected results in motion execution, especially in critical configurations, such as balance sensitive poses, where even a small error factor destroys the whole motion sequence.

Last, but not least, safety of the robot is every researcher's top priority. Robots are electro-mechanical devices costing thousands of Euros depending on

the type. Consequently, creating motions, not being able to predict the outcome, might ruin the robot. This the main reason, why all the more universities, companies, and researchers around the world spend time and money developing simulators that model robots, and their environments, as realistic as possible. In overall, having an adequate simulator, prevents the robot by executing unsafe motions, allows the usage of learning algorithms to improve motions, and saves time in terms of avoiding cross-compilation problems, uploading executables, turning on/off the robot, and other disadvantages existent in robotics research.

4.3.2 Methods In Motion Design

Robots are computers. So, why do we need robots in our life? The main reason, and distinctive characteristic between ordinary computers and robots is that the latter can perform actions and intervene in the physical environment they operate at. That's why motion design is of high importance in robotics. The primary goal of a robot having at least one degree of freedom is to move. Since the dawn of robotics, a lot of methods have been proposed to actually develop these movements, but all these are nothing more than satellites, orbiting around three fundamental ideas.

4.3.2.1 Forward Kinematics

Forward kinematics is the computation of the position and orientation of robot's end effectors as a function of its joint angles. Forward kinematics, is a mathematical approach by which you can calculate where the end effector is going to be, facing in what direction, after executing a simple step of a motion sequence.

By the term end effector, we can consider every joint, since every joint serves as an end effector of the chain containing all the previous joints. Given that we have an automated way of predicting the position of each joint, we can say that developing a motion can be fairly easy, and straightforward.

Unfortunately, even now that we have such a strong mathematical background, the development of simple motions in a robot with more than 5 or 6 joints is a tedious work, not handy in many cases.

4.3.2.2 Inverse Kinematics

Inverse kinematics is the reverse process of forward kinematics. Inverse kinematics is the process of determining the parameters of a jointed flexible object (a kinematic chain) in order to achieve a desired pose[1]. It is a more intuitive approach, in terms of how humans perceive motions. This will be clearer with an example.

Picture the scene of trying to grab a glass of water. Which are the first reflective thoughts popping out of your head? Of course, where the glass is, and in which direction should the hand approach it. Afterwards, some complex calculations are taking place, without even understanding it, and in a few milliseconds you know the right angles each "joint" should be positioned, in order to successfully complete this task. It worths mentioning, that the solution found is considered according to a cartesian space attached to a point that we are familiar with. For the previous example, we are sure that these parameters are not going to be according to the corner of the room (even if we could), but according to your torso for example. Another human advantage is the ability to recalculate these parameters if something changes in your environment (perhaps an obstacle between you and the glass).

Inverse kinematics serve the robotics research in great extent and is widely accepted by researchers, but once again as the forward kinematics, the mathematic background needed and the complex computations taking place, does not allow one to use them extensively in domains such as RoboCup.

4.3.2.3 Work-Configuration Space

The third solution is closely connected to the Inverse Kinematics approach, but is a lot simpler. When working in the configuration space, we are able to freely move the joints of the robot in the desired position, which is quite similar to Inverse Kinematics, in terms of knowing the target and the desired direction. The distinctive point is that in the configuration space you must also set the joints of the whole chain, and you are not going to calculate these angles analytically.

[1] Definition from www.wikipedia.org

This approach is fundamental and is being widely used in industrial high precision robots, where accurate, repeating motions are being executed over time. The only applicable problem to this approach, is the support of the robot manufacturer, to allow you set the stiffness of the joints to zero, and also have the ability retrieve the joint angles of the desired configuration.

Chapter 5

Kouretes Motion Editor

Have you ever wondered, why do we develop software? Of course you have, and the answer is simple; because we want to simplify our life. Machines' existence is to assist us on demanding tasks, which most of the times are physically impossible to be done, fullfilled, or maintained by humans.

5.1 Our Experience

Kouretes team has been taking part in various RoboCup competitions and exhibitions, making its debut at RoboCup 2006 event in Bremen, Germany. During the first two years, *Kouretes* team was spending most of its resources in the development of the vision and behavior module of the Sony Aibo robot. In the beginning, former team members were programming exclusively in an interpreted language, the Universal Real-Time Behavior Interface (URBI), without any use of existing code. In the subsequent years, we managed to use existing code of the SPQR's release, thus we continued working on vision and behavior topics.

More than two months were spent before discovering out that most of the motions already developed by other teams were not usefull to us, because any further manipulation or tuning was not possible. Even when we were improving the behavior code, our team was unable to improve its play, due to many factors. One could be for example the inadequate "ball grab" motion. We also found out that a slight variation in the field carpet or the joints of the robot, could easily deteriorate already developed and tested motions. Even small error factors to the

previously mentioned example results in a motion that rarely manages to actually catch the ball.

Our very first thought was to find a "great" motion editor. We started searching firstly inside the Robocup community, and then expanded to other research areas. To our disappointment, we did not find that "great" motion editor, but found instead very specific solutions targeted in specific robots, whose output most of times was difficult to be manipulated. Nevertheless, we had been working under these circumstances not very long before the Sony Aibo was about to be replaced by Aldebaran's Nao platform in the Standard Platform League. This change made us understand we were about to lose everything. Now,what?

5.2 Technical Knowledge vs Technical Skill

"Technical Knowledge can be considered the in-depth understanding of anything that can be applied or reasoned with, in any shape or form, for any issues or applications"[1].

On the other hand, *"Technical Skill"* is the ability to put in practice this technical knowledge acquired by any means.

According to these, we, as engineers, even though we had the technical skill, we took our chances and failed; we did not manage to acquire the most out of the technical knowledge produced in the past.

5.3 The Concept

Once again, we were to ground zero and had to decide where to start in order to solve the problem of designing complex motion patterns. The decision was to develop a client that would serve some basic functionality over the network communicating with the robot and should be strictly attached to some features. These features can be concluded in the following list :

- To be OS-independent, and be able to compile under Windows, Linux and MacOS.

[1] Anonymous' definition

- Detach the client from the robot and run it in the local machine, in order to reduce the execution load of the robot.
- Have a Graphical User Interface, that should be fast, light, simple, and intuitive.
- Communicate in a reliable, simple and fast way with the robot.
- Communicate with at least one well established, and widely used simulator.
- To be platform independent, and easily parameterizable for any given robot.
- To be easily maintained, by others.
- To use a simple, close to human language, communication protocol.

And the name of it...**KME** (3).

5.4 From Theory to Practice

The goal behind the development of KME , which stands for Kouretes Motion Editor, is to provide an abstract motion design environment for the common robot practitioner, which hides away the technical details of lowlevel joint control and strikes a balance between formal motion definition using precise joint angles in the configuration space of the robot and intuitive motion definition using manual joint positioning in the real-world work space of the robot. Such an abstraction yields a number of benefits, which served also as the driving force behind this work: (a) arbitrarily complex motion patterns can be easily designed without ever writing a single line of code, (b) motion patterns can be rapidly designed and tested through a simple and friendly interface, (c) motion patterns designed by one user can be easily shared, understood, used, and modified by other users, (d) various real and/or simulated robots can be accommodated simply by reconfiguring the backend of the tool, (e) resulting motion patterns can be used as seeds in learning algorithms for further fine-tuning, and (f) proprietary motion patterns could be reverse-engineered as recorded sequences of complete or partial robot poses and subsequently manipulated at will.

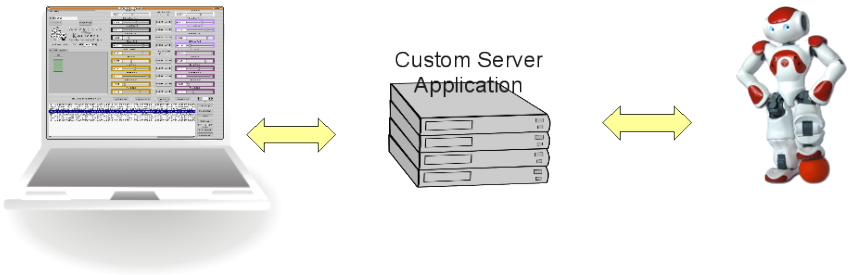

Figure 5.1: Client-Server architecture

5.5 In-Depth System View

5.5.1 Architecture

KME is implemented as a client-server architecture (Figure 5.1), whereby the client and the server sides are interconnected over a TCP/IP network as described below. The server is a special controller attached to the real or the simulated robot, in the sense that it is platform-dependent, resides either on the robot or on some machine connected to the robot, and has the ability to directly control the robot joints.

The server simply listens for a client on specific ports and undertakes the role of transferring joint values between the robot and the client, once a client is connected. The client is an independent software application running on the local or any remote machine and provides the graphical user interface (GUI) described

below. Communication between the client and the server is bi-directional; any set of joint values provided by the client can be transferred to the server and drive the robot joints to the designated pose, and conversely the current joint values on the robot can be read and transferred from the server to the client for storage and further manipulation. Note that other platform-dependent information, such as sensor readings, can be communicated between the client and the server, if necessary, with appropriate adaptation of both the client and the server.

Furthermore, several clients can be connected to and/or disconnected from a single server to facilitate access to the robot from multiple KME users, mostly as a means of enabling quick user switching and coping with network failures, and less for simultaneous motion editing, which is certainly not recommended. End-Users have the ability to develop the server in the way they want in order to provide the desired functionality. Some examples could be the PC-To-Robot, PCs-To-Robot, PC-To-Robots or PCs-To-Robots comunication (Figure 5.2). The abstraction offered by KME allows you to freely choose how to use it in collaboration with the custom developed server.

For maximum portability, it was decided to use open source software for the development of KME . In particular, the core client code is written in C++ and can be compiled under any popular operating system (Linux, Windows, MacOS) using a standard C++ compiler. The current graphical environment is based on FLTK (4) libraries, however the core client code is structured in a way that allows interfacing with other popular graphical toolkits, such as QT and Tcl/Tk. Finally, the server code for the Nao robot has been written in the interpreted script language Python for simplicity and portability. Our preferred execution mode is to run the server on a remote machine which communicates with the Nao robot using the distributed NaoQi architecture.

5.5.2 Networking

5.5.2.1 TCP/IP Protocol

All interprocess communication between the server and the client is based on the TCP/IP protocol. This protocol has been chosen, instead of using UDP proto-

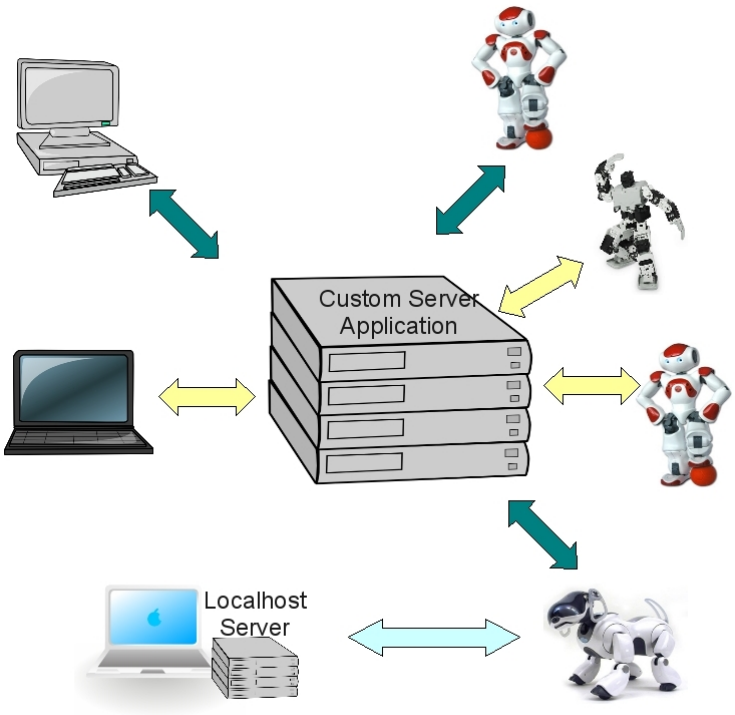

Figure 5.2: K$_{M}$E 's architecture provides maximum flexibility to the user

col for a number of reasons. TCP/IP enables cross-platform, or heterogeneous communication, but the distinctive characteristics that led us choosing it are:

- Good failure recovery
  Even when a part of the network fails, the rest network will be able to survive.

- High error-rate handling
  Re-routing of lost messages ensure that the message is going to be delivered sometime.

- Correct receiving order
  TCP/IP protocol, ensures that all packets are going to be delivered in the correct order.

Once started, a server looks for an available port beginning from port 50000 increasing the port number by 1 until an available port is found or a predefined limit is exhausted. From our experience, we have set that limit to 10, because during the development period not even once did we find the first free port to be greater than 50003. If a port is found, the server listens for clients connecting to that port. When started, the client must use the port number posted by the server to establish a connection.

5.5.2.2 Types Of Messages

All messages exchanged over the network use simple ASCII-based structure for maximum portability purposes. Most messages contain complete robot poses, but there are also other smaller messages, for example for connection initialization, setting joint stiffness values, or communicating optional sensor information. The largest message communicated between client and server has a size of 144 bytes and corresponds to a complete robot pose description consisting of 22 signed floating-point numbers with 5 or 6 ASCII characters per number ([-]`x.xxx` format) separated by a marker (%). The exact types of messages, are going to be presented below, and is very important for potential users to keep this structure always in mind, especially when you want to create a different server (see Appendix B for more information). The reason of creating a different server than the one provided, is primarly the use of KME with another robotic platform. In order, to help the reader with the development of his or her server application, we are going to present the simple protocol used by KME :

disconnect

The server should end the communication in a **clean** way, closing all sockets.

load

The server should retrieve all the joint values, and then send them back to KME .

stif_on%x.x

The server should set the robot's stiffness on to the desired value x.x, with 1.0 being 100% of the availabe stiffness.

stif_off

The server should set the robot's stiffness off, in order to manually manipulate the joints.

play%[−]x.xxx%...%[−]x.xxx%end

The server should drive the robot joints to the desired position. The number of the joints may vary from robot to robot, but in the Aldebaran Nao robot, this is 22. The exact message each time can be previewed in the browser field of KME . It should be noted that the server can also work when a replacement of "play" to "pose" is taking place. Some time ago, during the development process, in order to distinguish the poses sent to the robot using the "Step Motion" or "Play Motion" buttons from the ones sent by moving the sliders, we used "pose" instead of "play" for the latter. For debugging purposes, we continue to use it.

Besides those messages, when it comes to the Aldebaran Nao robot, there are some extra messages that can be used, in order to have greater functionality. These messages are:

x_y_stif_z

The server decodes this message in the following way; "x" can either be "l" or "r" representing left and right respectively, "y" can either be "arm", "ankle", or "leg", and z can be "on" or "off". By this message the server should set the desired stiffness to the desired joint chain. An example of this message could be :

```
l_ankle_stif_off
```

which will set the ankles's stiffness of the left leg to 0.

talk%This is a test%Volume%50.00...

The server receiving this string is going to send the string "This is a test" to the robot, and also set all the parameters with the corresponding value of each parameter.

getRobotState By this message, the server sends the robot's state information to the KME , in order for the latter to visualize these data. Right now, only the battery state is used, but the main reason, we wanted this functionality, was to visualize the servos' temperatures, since this information is valuable in motion design. Unfortunately, by the time this thesis is being written, Aldebaran provides only a simulated joint temperature, which is not accurate, and at least for now it is pointless to graph that information.

Our top priority was to have a reasonable overhead in the robot while developing a motion and also achieve real-time communication to a logical extent. Messages are being exchanged over the network only as needed; the client talks to the server only when the robot pose set by the client changes and symmetrically the server talks to the client only when the current pose on the server side is requested by the client. It is quite important to understand that communication must meet real-time constraints given that any change on the client side must be immediately reflected at the robot joints on the server side. The requirement of minimized latency is dictated not only by efficiency goals, but also by safety concerns; a motion executed on the robot several seconds or minutes later than expected due to a network freeze or congestion may have fatal consequences. Under the current communication protocol, the server and the client run smoothly and transparently in a robust and predictable way even over the wireless link between the robot and the computer. In the frequent (hopefully...) case, that the reader wants to develop his/her own server, please consult Appendix B for more information.

5.5.3 The Graphical User Interface

The KME graphical user interface (GUI) provides the means for the creation, capture, management, storage, reproduction, and export of any sequence of robot poses. The entire GUI consists of three components as shown in Figure 5.3.

5.5.3.1 File-Connection Component

The component on the left-hand side can be divided into three areas. The menu on top left contains the menu for file operations. The available options are :

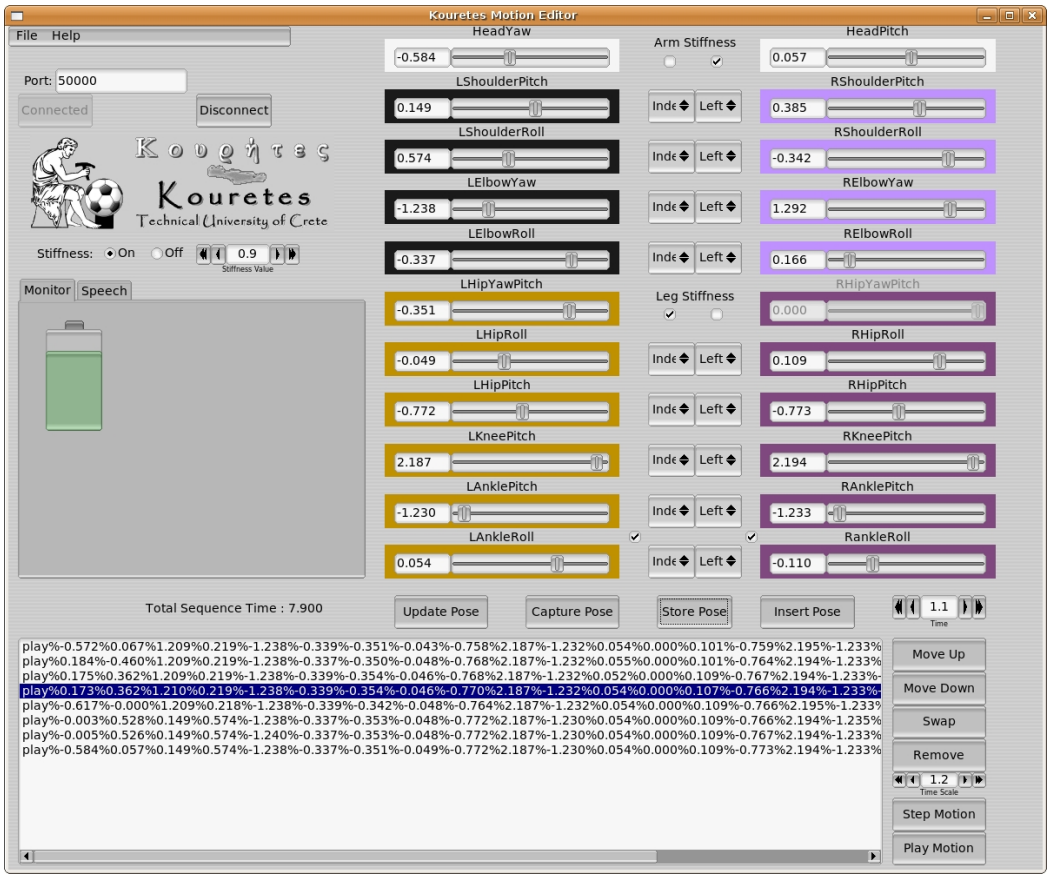

Figure 5.3: Kouretes Motion Editor, Graphical User Interface

Export to .kme

Exports the designed motion in a .kme file. There is also support not only for checking existence of the same filename, but also overwriting the file when the user wants so.

Load...

Opens an existing .kme file. The user can then edit it, or generally manipulate it; we can then continue working in the KME program, having available all the saved poses.

Quit

Quits the KME program.

Help/About

A thank note and contact information for the developer.

In the bottom field, form and buttons exist for establishing a connection to a server using a specific port number. There is also the possibility of entering a different IP of another machine if KME and the server are in distinct machines. In that case, in the port field one should write for example "192.168.1.10:50000" instead of "50000". Below is the radio button for turning joint stiffness off and on, and also set the desired stiffness value indicated in the counter widget.

Finally, a tabular display is available for visualizing optional platform-specific sensor information. In the Aldebaran Nao robot, we have two available tabs, one working as a monitor, which for now only serves the battery information and another one dedicated to speech properties. As you can see, in Figure 5.4 a variety of options concerning the speech module are available. Options include a drop-down menu, which defines the voice we want to simulate. To this date there is only one available voice from the Aldebarans' ALTextToSpeech module. A volume slider is available in the range of 0 to 100, giving the percentage of volume. Some extra parameters such as the Pitch Shift, Double Voice, Double Voice Level, and Double Voice Time Shift also exist. We also give the freedom to the user not only to play the text from the input box, but also save it in a .raw file. Finally, there are two buttons, "Talk" and "Save", that allows the user to hear the speech real-time and to save it in the browser as an instant of the

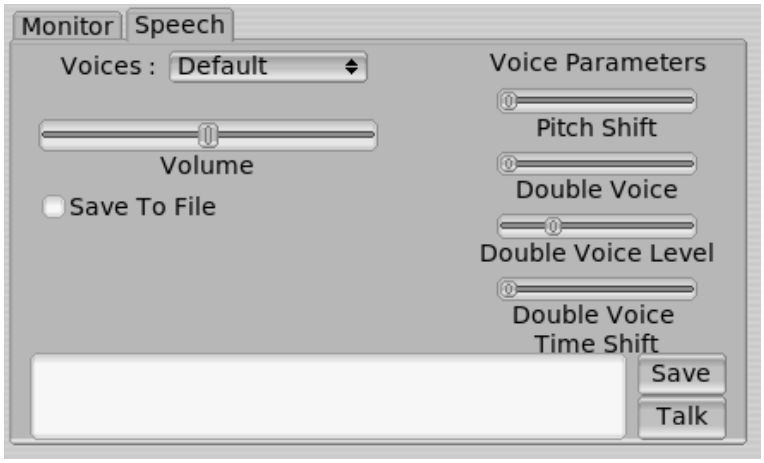

Figure 5.4: Speech tab allows for manipulating speech to text options.

desired text plus the parameters defined by the user respectively. The last option allows for developing simple behaviors, since the robot can sequentially execute some poses and use Text to Speech module before, between or after them. More information concerning all the uncleared details, can be found at Appendix A.

5.5.3.2 Sliders Component

The component on the right-hand side offers 22 sliders, one for each joint of the robot; two of them (L/R HipYawPitch) are coupled together by default. Notice that the layout of the sliders resembles the location of the joints on the Nao robot and each joint chain (arm or leg) is marked by a distinct color. The user can set the value of any joint either by sliding the corresponding slider to the desired position or by directly setting the desired arithmetic value. Any change made to a joint value is immediately communicated to the robot through the server (if stiffness on the robot joints is enabled), therefore the current robot pose coded by the slider values is always reflected on the robot. Symmetric joints, for example LShoulderPitch and RShoulderPitch, can be optionally coupled using the corresponding drop-down menus to select the type of coupling (matching or mirrored configuration) and the dominant joint (left or right). This feature is useful for creating symmetric or anti-symmetric motion patterns. A series of check buttons can be used to turn on/off stiffness locally on specific joint chains

to allow for designing of motion patterns using only a single joint chain. These chains are, Left/Right Arm, Left/Right Leg, and two virtual chains in the ankles. Independently, setting the stiffness off, in one or both ankles, is very usefull, since every time, you are sure that you achieve the largest available contact area, between the feet and the ground.

5.5.3.3 Pose Sequense Editor Component

Finally, the component at the bottom is a robot pose sequence editor. The sequence of poses can be indirectly edited as needed; poses can be inserted to or deleted from the sequence, they can be swapped with other poses, and they can be moved up and down to the desired place in the sequence. Furthermore, there are available buttons for playing step by step the desired motion but also for executing the whole sequence.

5.5.4 Motion Design

Any complex motion pattern created using KME is a timed sequence of robot poses in the configuration space. Robot poses can be created either by setting joint values through the sliders of the GUI or by capturing the current joint values of the real or simulated robot. The current robot pose coded in the sliders can be captured and stored at the end of the sequence using the "Store Pose" button or right after the currently selected position using the "Insert Pose" button . The "Update Pose" button can be used to update the currently selected pose with new values. Alternatively, for pose generation the user may manually move the robot joints to any desired configuration (under no stiffness) and use the "Capture Pose" button to capture the current joint values. The user can also adjust the transition time between subsequent poses, which implicitly determines the speed in the motion of each joint. Note that the time value stored with each pose is the transition time from the previous pose to the current one.

Finally, the user can "play" the current pose sequence from any point (either in a step-by-step fashion or continuously) to observe the complete motion pattern on the robot. Once the desired movement is complete, the pose sequence (or its symmetric one, according to the sagittal plane of the robot body) can be exported

to a file and can be further used within any robot controller by simply invoking a motion execution routine.

Designing motion patterns using KME can become a lot more interactive, as we discovered along the way. Consider two consequent robot poses A and B. The user may want the robot to move from pose A to pose B in a certain amount of time, however this may not be possible because of limited acceleration or mechanical load constraints. Using the step-by-step execution, the user may try to play the pose transition from A to B, however the robot may end up in some otherpose C under insufficient time or under mechanical load stress on the servos. Instead of trying to fix pose B (or A), the idea is to capture pose C from the current joint values; the transition from A to C is clearly a safe one. The design of the remaining motion pattern begins now from C and may shoot either for B (if possible) or for another target robot pose. Building the motion pattern in this iterative manner yields a motion sequence which complies with time and load constraints.

5.5.5 Motion Execution

Motion patterns designed using KME can be subsequently incorporated and reproduced within any robot controller without requiring the presence of KME itself. This is accomplished using a simple C++ routine which simply executes the stored motion patterns. In particular, the KME files found on board the robot are loaded into the main memory during initialization. A call for motion execution specifies the name of the desired motion pattern and a time-scaling factor (a real number around 1.0). The executor routine retrieves the specified pose sequence and executes the poses sequentially using a linear interpolation between them to avoid motion jerkiness. The time-scaling factor is used for speeding up or slowing down the execution as it multiplies the time values of each pose. A value of 1.0 corresponds to the nominal stored execution time; the user may optionally export the scaled or the nominal time values in the motion file. It should be noted that the motion executor is a simple open-loop scheduler;unexpected and/or uncertain events should be handled by a higher-level behavior module.

5.6 Simulators

5.6.1 Webots Simulator

One of our first thoughts were to make Kouretes Motion Editor available to communicate with Webots (5) simulator as well. Practically, this is a great opportunity for the robotics researcher to try some of the motions already designed on the real robot, or even design them from scratch. We have to admit that motion design is not as fast as with the real robot in terms of the development time, because the user does not have the ability to move joints by "hand" (by clicking on the desired joint); at least for now. Nevertheless, you have the ability to work with the sliders, which is a lot faster than working with the Webots motion editor. What is great with Webots is that the physics engine is working quite good, thus you can use the real robot's .kme files to test them in the simulator and vice versa.

Webots' nature does not allow for stiffness operations, and in most cases you will not need it. What we care more about in a simulation environment can be the physics control. Thus, if you have PRO key, you can modify your server and use the stiffness radio button in order to set on/off the physics of the simulator.

For now, we provide you a server compatible with NaoQi, which is identical to the one for the real robot. Thus, if you want to use KME with Webots, you must have a Webots PRO license to acess NaoQi. In order to avoid this narrow option and broaden KME 's users, we are now developing and will soon release a client that will work with the Nao robot without Naoqi and the need of the PRO license. This is mostly intended towards people that want to try KME and also others participating in the Robotstadium competition. We hope you find this feature usefull in your work and furthermore, it is a great way to continue working, during a flight...

5.6.2 Microsoft Robotics Studio

Considering we are using NaoQi in order to communicate with the Webots simulator, we "believe" that KME will also work with the Microsoft Robotics Studio(

MSRS). Unfortunately, no tests have been made yet, but probably we will provide you with an MSRS server in the near future.

5.7 Custom KME

KME can be customized theoretically for every robotic platform available. What should you do from your side is to create an XML configuration file, which has to represent the robot's joints, ranges, and other needed information by KME .

KME 's graphical interface is dynamically created by loading this XML file. One part of the Nao robot's XML file is shown here:

```
<Robot>
        <manufacturer>Aldebaran-Robotics</manufacturer>
        <type>Nao</type>
        <JointNumber>22</JointNumber>

        <Joint>
                <name>HeadYaw</name>
                <minBound>-120.0</minBound>
                <maxBound>120.0</maxBound>
                <step>0.0</step>
                <color>24</color>
                <coupledWith>None</coupledWith>
                <couplingType>None</couplingType>
                <indexOrder>0</indexOrder>
        </Joint>

</Robot>
```

The Joint block is being repeated for every joint available in the custom robot. Ranges are in degrees and all fields are compulsory in order to run KME without flawlessly. Please note that for now, we do not have an XML validator, thus you should be carefull when writing your own XML files. If you face any problem, feel free to send us an email with the XML file attached to help you resolve this issue. Detailed information on the server can be found on Appendix B, while the XML's explanation and structure can be found in Appendix C.

Chapter 6

Results - Empirical Evaluation

RoboCup's nature is so different than anything else scientists may ever attend. RoboCup is a very competitive environment, where most researchers benchmark their work in real-world problems. Consequently, all our proposed solutions were to be tested and evaluated during this competition.

6.1 KME at Robocup

KME has been successfully employed by *Kouretes*, the RoboCup team of the Technical University of Crete, during the RoboCup 2008 competition which took place in Suzhou, China in July 2008, for designing various special actions (stand-up, ball kicks, goalkeeper actions, bending of body).

Under carefull inspection and considering the weaknesses of the Nao robot, *Kouretes* team had a number of motions created and optimized a week before the competition. To our expectation, most of the motions were inadequate, or even useless, since everything was optimized in our lab, using our own carpet. In some cases we had to redesign a motion, or develop it from scratch. This was the chance to prove KME 's usefullness. One representative example was the usage of KME in the design of a stand-up motion (Figure 6.3) in reasonable time.

6.2 Soccer Skills Developed

A number of motions were much-needed in a soccer game. In order to develop a stable, quick, and safe motion for a robot, one must consider all the special characteristics of the robot. These include servos' speed, maximum torque available at each joint, and a lot of other factors. All motions presented below, are explicitly developed for the Nao robot and cannot be transported to another platform, at least in an intuitive way.

6.2.1 Bend

Aldebaran Nao robot in its debut was in its second version (V2). As mentioned on Chapter 3, this version was equipped with one camera only making it impossible to see the ball from the zero position[1], when the ball was in the short frontal area.

Development of the bend motion is rather trivial, but has to be optimal; these kind of motions demand enough torque to support its body weight. Robot body can be considered to be divided into two masses. The first to be the upper body from the head to the hips and the legs to be the second. Robot in that position, transports most of its weight keeping balance to both left and right HipYaw joints, as shown in Figure 6.1. Even though, this pose does not seem to stress the joints, precautions should be taken to assure the development of the safest motion. Repeated execution of the same unsafe motion during a game, will eventually break the robot, especially when it comes to Nao V2.

This motion consists of three poses. The first and third are identical and represent the starting and ending points. These can be the Zero Pose, but most preferable is to use another pose which must be the starting and ending point of walking pattern. The intermediate pose, is the one where the robot bends its torso.

There were two options in the development of the bend motion. On the one hand, keeping the knees straight, as shown in Figure 6.1, and on the other trying

[1]By zero position, we mean the pose, where every angles' value is zero

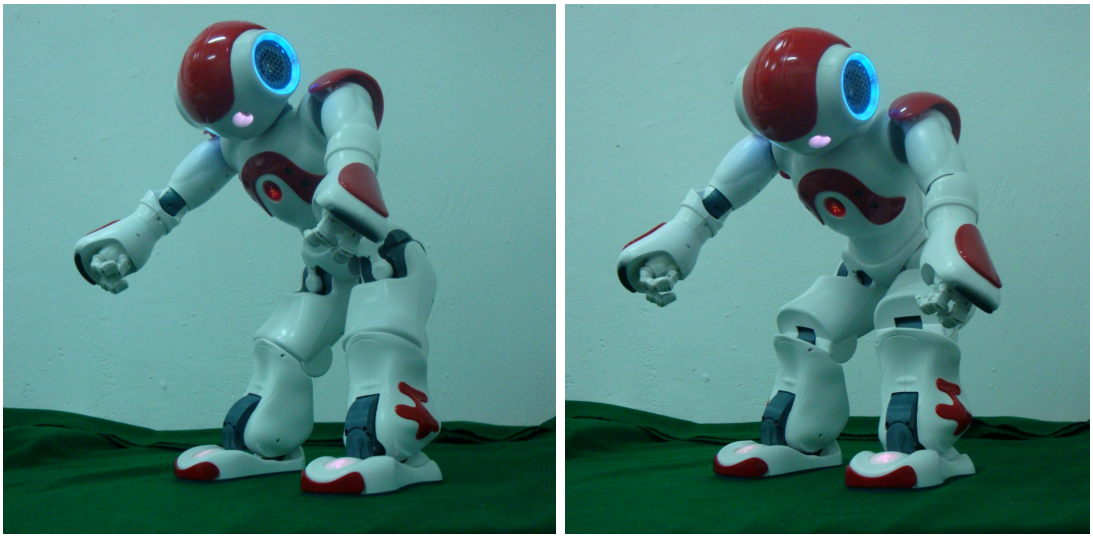

Figure 6.1: Most of the stress is transported to the hip joints. Forcing the knees to bend at the same time, we achieve more stability.

to bend the knees as well. What we came up with was that bending the knees, was not stressing the joints that much and was more stable in the end.

6.2.2 Stand Up Motion

A much-needed motion in the SPL-Nao league was that of standing up after a fall. Well before the first attempts on the real robot, one stand-up motion was developed in the Webots simulator. The routine on the simulation was based on the hands of the robot, but it was discovered that the servos on the Nao arms are quite weak to support its body weight, therefore one needs to carefully move most of its weight to the legs for a successful stand-up procedure.

6.2.2.1 The Three Phases

Stand up motion in the V2 robot, due to its complexity can be divided into two phases. Before considering that the title is misleading, there exists a third phase which is the first one to be executed. This is the motion that turns the robot from a face-up to a face-down position. Unfortunately, we were not able to develop a

stand up routine from a face up position. Nevertheless, the two most important phases are to bring the robot from a face-down position to the embryo stance having its feet in full touch with the ground. From that position the second phase is a lot easier, since the only thing the robot has to do, is to move its weight from one leg to the other and gradually acquire the standing stance.

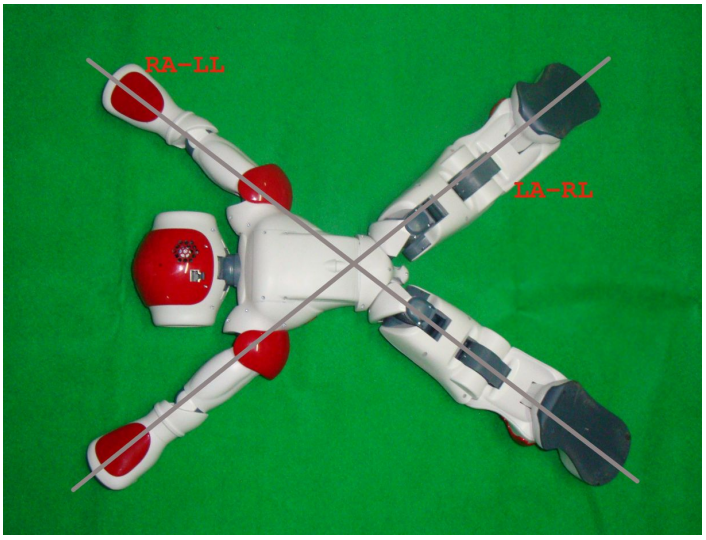

Figure 6.2: Symmetries found in the robot in the coronal plane. Left Arm - Right Leg and Right Arm - Left Leg, help release the stress from the other joints

As mentioned before, the weakness of the Nao's arms have set a hard constraint that forced us to work in the balance of the robot. Even when using a sagittal symmetry in humanoid robots, when it comes to the stand up motion, most of the work is done in the coronal plane considering as starting point the face-down fall. We hypothesized two straight lines, intersecting a few centimeters above the hip joints of the robot, or in the lower torso (Figure 6.2). These lines are the symmetries allowing the robot to balance its weight in the **L**eft **A**rm - **R**ight **L**eg (LARL), or the **R**ight **A**rm - **L**eft **L**eg (RALL) formation. Using in every step the first or the latter symmetry we were able to move the free chains to execute the desired motion.

For example, when the robot is crouching down on all fours, using the LARL formation we can freely move the right arm and achieve any desired position,

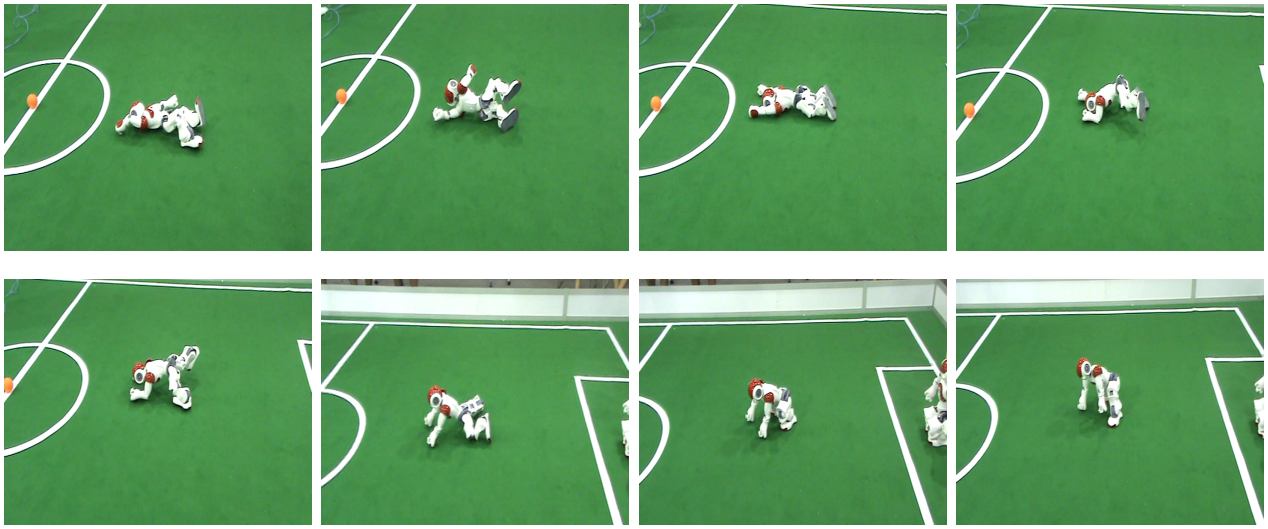

Figure 6.3: Stand up motion, presented in RoboCup 2008, Suzhou, China.

as long as the left leg is positioned in the proper stance. In order to help even more the weak arm joints, the opposite leg can be raised in the air. Thus, torque applied to the robot is maximized and the latter takes advantage of that in order to reduce the stress applied to the moving joint. Figure 6.3, presents the key poses of the whole sequense of poses.

Finally, the creation of a special action module was inevitable. Using the inertial sensors of the robot, the agent was able to determine the orientation of the robot body after a fall and execute the appropriate motion that first bring the robot to a face-down position. Then it attemps the stand-up motion. Special parts of code were written to sense whether the robot was fallen in a side position.

Surprisingly, this stand-up motion designed on the carpet at the home laboratory did not work on the carpet at the competition venue. Thanks to KME , it took only about 30 minutes and two people holding the robot to design from scratch a new stand-up motion for the new carpet.

6.2.3 Kick

One might think kick is the most important motion in the RoboCup Soccer Competition. This is partialy true, since past competitions have shown that this is not the rule. In 2008 competition for example, the winning team NUManoids, in the finals won by walking towards the ball in the penalty kicks period, while GT/CMU team spent valueable time aligning according to the ball and the goal in order to kick. Nevertheless, what most teams pursue is to have the best kick not only for the obvious reason, but also to increase the prestige of the team. Kicks are being improved all the time in terms of speed, stability, and precision. It can be considered more or less as a race between teams.

6.2.3.1 The Four Phases

Once again the kick motion can be divided into phases, but now into four. The distinctive characteristics differentiating the kick motion by other soccer skills are mainly two. The first one is the transition from a double legged support mode to balancing in one leg. Complexity of creating closed loop kick motions discourage most of the teams including us to develop dynamicaly stable kicks and follow statically stable motions. Thus, in the first phase, the robot leans to the supporting leg either by using predefined functions created by Aldebaran, or by using a reference pose created in our labs using KME . The latter now, can be either a hand-crafted pose, or the one reverse engineered by the Aldebaran's motion. Both work pretty well.

In the second phase, the free leg can be raised from the ground and moved some centimeters back. Depending on the kick type, in the fourth phase, the kicking leg is brought in the frontal area to hit the ball in the desired angle, height, and speed (indirectly by setting the duration of the motion).

The two subsequent phases, are the one that the robot brings the free leg back and aligns it with the supporting leg, and the final that returns to double leg support mode again.

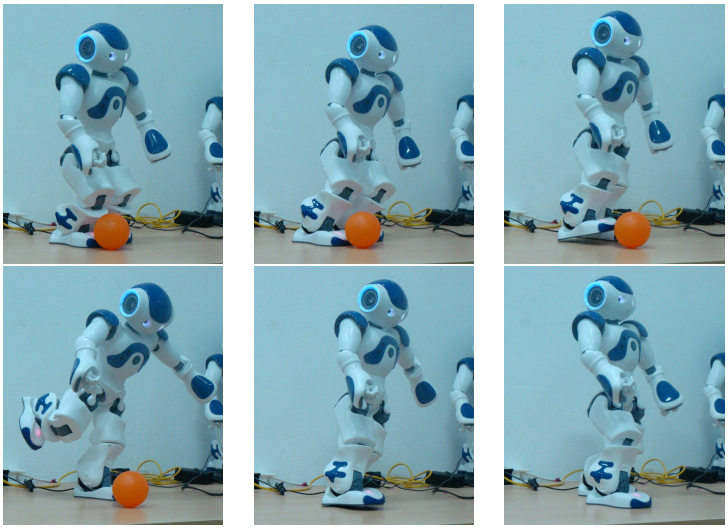

Figure 6.4: All the key poses that consist the kick motion

6.2.3.2 Points To Consider

Developing a kick is not an easy task, but in most cases one does not have to redesign this motion for other types of carpets. Generally, while designing a kick motion, keeping static balance in every pose must be ensured and most importantly that the transition from one pose to other is soft using small pauses between poses to eliminate vibrations by de/acceleration on the joints. Of course "soft", is not applicable to the kick itself, since the robot has to bring the leg from back to the frontal area, as fast as possible to maximize the speed of the kicked ball. Thus, the researcher strives to keep a balance between speed and stability.

More work can be done using the robot's hands to improve balance. In that field the expansion of arms is a good start to imitate human kick. For example, when having a right kick, the support leg is the left. In the start of the motion, we expand our left arm in the left frontal area, while bringing back the right. In the kicking phase the left arm, comes back and the right front twisting the body in order to give more power to the kick. While this motion works perfectly for human bodies, in robots, twisting is not an acceptable behavior, due to stability

Figure 6.5: Goalkeeper dive. Adding the left dive results in a 75 cm covered area.

and localization issues. In order to avoid it, we didn't use the arms extensively.

In the RoboCup 2008, we were one of the few teams that had a kick motion and one of the fewer that this kick was powerfull enough. In Figure 6.4 you can see the kick we used then modified now using KME to get even better results.

6.2.4 Goalkeeper Dive

The new Standard Platform Nao League rules have extended the size of previous year's goals, since the new robot is bigger. However, the width of the standing Nao was pretty much the same with that of an Aibo. To make some comparisons, Nao's guard area when standing is around 25 cm, when at the same time Aibo's area is 20 cm. The goal's width is 130 cm and the Aibo's goal was 80 cm.

This difference can be more obvious using percentages. If we have the Nao standing in front of its goal, then $\frac{25}{130} = 19.23\%$ is covered by the robot, while in Aibos $\frac{20}{80} = 25\%$ of the area is covered.

This difference yields more problems for the goalkeeper. Considering as well, the fragility of the robot and the time needed to recover from a dive, saving a

kick by falling in the floor was out of the question. So, the decision was made to take advantage of the long Nao's legs and place them in a wide open position; being carefull not to allow the ball to pass between its legs (Figure 6.5). This motion or its symmetric one according to the sagittal plane, can cover an area of 50 cm covering around 38.46% of the goal while the Aibo's dive covers 50% of the area. Nao's dive does not seem to be good enough, but considering its symmetric motion that changes. In the frequent case that the robot is standing, we can choose the type of the dive we want to perform. Subsequently, the area that can be covered from the very same starting point can be 25 cm if it stays still, 50 cm diving at the right(Figure 6.5), and 50 cm diving at the left side. That results to a total of 75 cm covering area (the 25 cm in the center overlap), which is around **57.7%** of the total goal area.

Goalkeeper's behavior was the only one shown in Robocup 2008. This and the stand up motion were those that received the most flattering comments by other teams.

Chapter 7

Related Work

KME offers some innovative ideas, however it also bears similarities, which are mainly based to intuitive ideas found in other existing tools for designing complex motion patterns. To find a "competitor", is not that easy, because every tool has different characteristics to offer. Choosing the right tool will be based on the type of the robot one wants to work on and also the purpose of usage. Some discussion below on each tool available on motion designing will help the user find the tool that meets his needs.

7.1 Choregraphe

Choregraphe (6) is a proprietary software package developed by Aldebaran Robotics to facilitate complex behavior programming on the Nao robot. Its beta release came out in June 2008, well after the development of KME , but the functional release came out only in January 2009. Choregraphe offers a cross-platform environment that allows the user to build various movements and behaviors on a real, a simulated, or a VRML model of the Nao robot. To this end, it combines time-based and event-based approaches. Time-based design is used to schedule motions and multimedia material over time. Different timelines can be used depending on the current execution context. An event manager is responsible for identifying the current context based on the occurrence of events and triggering the appropriate behaviors. Choregraphe's capabilities are well beyond KME 's capabilities, which were not to be included in KME .

KME is a tool dedicated to the RoboCup environment and has nothing to do with behaviors in terms of having event-based functionality. Choregraphe is not customizable and comes with heavy system requirements for reasonable real-time performance. Furthermore, the cost of Choregraphe can be considered prohibited for end users, since it can reach the 1/3 of the robot price. RoboCup teams only use a portion of Choregraphe's functionality, thus the cost is high.

7.2 Motion Designer

Motion Designer (7) is a tool developed by the RoboCup team Microsoft Hellhounds and comes in two versions, one for the Aibo and one altered for the Nao robot. Motion Designer allows the design of complex motions interactively with the real robot by offering several manipulation functions on timed sequences of poses and a custom transition editor for combining different motions, which is intended only for their code. The whole concept behind Motion Designer is closely related to KME , but firstly does not support simulated robots, and furthermore is neither freely, nor commercially available, so we have not worked with it to have a more thorough opinion on this tool.

7.3 MEdit

The MEdit tool (8) was developed by Sony for the popular Aibo robot featuring a total of 20 degrees of freedom. MEdit is a graphical tool that allows users to generate complex motion patterns as sequences of poses. Unfortunately, its first (and only) release was incomplete in terms of functionality. While the design of motion patterns can be done extremely easy through direct setting of joint values or through manual joint positioning on a VRML robot model, export and integration of such motion patterns into generic robot controllers and applications is rather cumbersome. This happens because each pose is saved as a separate file and the motion is uploaded to the robot only through the custom-made R-Tool and triggered only through the custom language R-Code. Nevertheless, it was a good tool, abandoned quickly not allowing community receive the best out of it.

7.4 Skitter

Skitter (9) is another motion editor for Aibo robots with more capabilities. It allows combinations of motions (20 degrees of freedom), lights (32 independent LEDs), and sounds (MIDI and WAV playback) along a time line. Such combinations are called skits and can last up to 4 minutes. Skits are designed using plots over time and a VRML robot model which can be manually positioned for capturing poses. The resulting motion patterns can be exported for use on the real robot, again only through R-Code.

7.5 Comparison

MEdit and Skitter are tied only to the Aibo robot and do not support direct interaction with the real robot. MEdit, Skitter, and Motion Designer are available only for the Windows™ operating system. Apart from these differences, the distinguishing feature of KME missing from the tools mentioned above is the ability to directly interact in real-time with the real (or a realistically simulated) robot and design safe and robust motion patterns in an iterative manner, as well as the flexibility in incorporating the resulting motions in a variety of robot software architectures. It is important to stress out the fact that working directly with the real robot overcomes several difficulties occurring when motion patterns designed on a VRML model without sense of physics and/or mechanical, dynamic, and kinematic constraints do not yield the desired effect on the real robot. KME currently focuses only in motion, ignoring light and sound, which really fall outside its scope and purpose. In summary, KME complements the existing tools by providing an alternative, flexible, effective, interactive, and customizable motion-design tool, when motion alone is at focus.

7.6 What KME Offers

KME was meant to be from the very beginning a tool that would help the RoboCup team *Kouretes* develop complex motion patterns. During the development process most teams didn't have a motion editor tool to create such motions

and they could not even find a free one. In order to support the community and the newer teams that just entered RoboCup after 2008 competition, we decided to provide KME for free. Teams are working under different programming environments in different operating systems and that's why it is multiplatform.

KME cannot be considered the best available tool and none could say that, since every tool has been developed for a purpose. Our goal was to help the researcher abandon the idea of playing with the angles. In the end of the day, the aim of this competition is to play soccer and not to have the best kick. So, create a good kick, and spend most of your time in Vision, Behavior, Localization etc. In case you insist on developing motions, then use the abstraction provided. Another option is to easily share, and understand the .kme files. Every team can create a motion and distribute it to others. I personally find it interesting, while some might argue on that.

7.7 What KME Does Not Offer

As its name states, KME is a Motion Editor and nothing more than that. You cannot build a behavior using event based approaches, but you can always create a series of motions including speech and in a way name it behavior; this is not an intelligent machine interacting with its environment. KME was meant and still is a supporting tool to extract the desired motions in reasonable time and easily adapt these later in the behavior development.

7.7.1 KME Outside RoboCup

No sooner than having developed most of the core code of KME , Aldebaran announced that will also provide the Academics version of the robot in the robotics market. Having some differences between those two versions, a simple question yielded immediately. Will KME be abandoned? The answer was no and the usage of a simple XML file format instead of using a static GUImade KME fully customizable for every robotic platform.

This was mainly done to help other universities use KME in their laboratories by simply creating an XML file based on their robotic platform. Now they can

have a fairly new motion editor for their robot, as soon as they have developed the intermediate server, between the robot and KME .

The third group we are interested in, are enthusiasts who have bought robots for entertainment. Should someone want to develop complex skills and don't want to use, or buy proprietary software then KME can be considered a great alternative.

Chapter 8

Future Work

Even though, KME fullfills all of its firstly set expectations, there are always things that can be added to improve this work in the future. Short-term plan include the addition of several bells and whistles to the next release of KME .

8.1 Porting KME to Qt

Developing KME using FLTK graphical library was not the best choice. We simply started working with this library, because prior evidence had shown it was stable, and some members of the Kouretes team also had prior experience as well. These reasons were enough to choose it. Unfortunately, two new parameters made us regret of using FLTK. The first one was the release of FLTK2, which was not compatible with FLTK v1.1.9 used in KME , and also till now, the most recent release of the second version is unstable. The other reason has to do with FLTK's usability. We have to admit that for simple tasks and GUIs FLTK is great and really fast. Nevertheless, when it comes to most recent approaches in human - computer interaction, FLTK is inadequate. Of course, there are some that support it and in fact prefer it against other graphical libraries; I don't.

Porting the code in Qt is the top priority for future work. The interface will become more beautiful, and take advantage of all the innovative ideas Qt offers. Finally, maintainance of KME will be easier for future members of the team.

8.2 Partial Configuration

One of the main characteristics of KME is the usage of whole body configurations, not allowing the user to capture partial configurations. This can be detoured for now by coding an intelligent motion executor that will each time use the desired joints. What we wish to fullfill is the support for partial configuration manipulation and/or execution over selected subsets of joints.

8.3 Motion Safety And Feasibility Analysis

In recent years, research from motion control has more or less moved into new areas, since basic control problems have been solved and other disciplinaries are working on these fields. Motion safety is the new feature should be imported into KME . Robots are still expensive devices and repairing them is expensive as well. Final users cannot do more than basic checks on the robots and in case of failure sending the robot back to base, or bringing a technician to the spot increases the cost and it is time consuming also.

Consequently, having a safety check while creating motions is a much-needed add-on, but this work in not very easy to be done. At first, plan is to use a threshold which would be the maximum speed a joint can move. If the user enters an invalid time duration between two poses then KME , would have to set that time to the minimum available, according to the maximum speed.

Further research would result in the feasibility analysis of each motion, and whether this motion can be executed by the robot. In this area a lot of work is required, and one branch of it can be the volume analysis of the robot, and transit our model from a skeletal one to a more realistic model, were collisions will be our primary goal to be investigated. Another outcome of this work would be to allow constrained motion planning for interpolating between poses in the motion sequence using arbitrary criteria. Of course this work is long-term and lies into fields far beyond KME 's current capabilities.

Additionally, we plan to release configuration files and server code for the Sony Aibo and the Robotis Bioloid robots. A major future step would be to allow for manipulation of various control structures (events, loops, branches, sequences,

etc.) over robot poses with the goal of designing simple closed-loop behaviors. Since robots are being increasingly popular in family entertainment, such tools will be undoubtedly important for the non-professional users.

Chapter 9

Conclusion

Conclusions are meant to indirectly point out the end of something. When it comes to KME this is totaly different, because this work is going to be published and not only help, but also contribute along with the work of other researchers in the RoboCup community.

9.1 Simple, but...

In a few words, we need a solution to a problem; motion design in robots featuring large number Degrees Of Freedom. This approach can be considered simple, but to the extent we searched none other solution was found to be applicable in that problem. Thus, the creation of a custom tool making it as generic as possible to avoid reinventing the wheel, was an one-way road.

9.2 What We Expect

My personal hope is to make KME freely available in order to receive as much feedback as possible. It is most preferable to get negative feedback to improve KME rather than abandon it. Starting from *Kouretes* team, this tool will surely accompany our work in the 2009 competition held in Austria. Furthermore, the participants list has been enriched this year and hopefully fairly new teams now beginning to develop their code in the Nao robot, will acquire KME and use it along with their behavior code.

Additionaly, KME can be considered a motion design tool applicable to a robotics laboratory in a university as well. In any of the above cases, if not in both, I will be satisfied to provide my help in a neuralgic domain, such as robotics science.

9.3 What Should You Expect

At this very point, KME in its stable version is normally used by *Kouretes* Team in the development of every motion. KME will be maintained and improved to the extent *Kouretes* team can in order to meet your need all time as community's demands increase.

Kouretes Motion Editor is released as is and *Kouretes* team does not pledge to offer any support. Nevertheless, I personally will be more than happy to hear from you any improvements, suggestions, questions, problems, and modifications you would like to see in KME .

At last I would like to thank all of you, who spent some time trying KME .

Appendix A

KME Manual Pages

A.1 The GUI

The Graphical User Interface of KME is shown in Figure 5.3.

A.2 The Menu

KME 's menu supports basic operations:

File/Export to .kme

This option saves the currently developed motion in a file using a filename specified by the user. Its shortcut is CTRL+S. A check question appears before saving the file, if another file with the same filename already exists in the destination folder. A recently added functionality asks the user whether to export the developed motion as is, or its symmetric one with respect to the sagittal plane of the robot (currently available only for the Aldebaran Nao robot), or its reverse with respect to ordering and timing. If a time scale factor other than 1.0 has been chosen, then KME will automatically concatenate its value to the filename, so that variations of an original motion are saved in distinct and easily recognizable files.

File/Load...

This option will pop up a file browser and will let the user choose a `.kme`

file to be loaded. Its shortcut is CTRL+O. Please note that if you already have some poses in the browser, these are not going to be deleted, and the poses from the file will be appended at the end.

File/Quit

This option simply quits the program. Its shortcut is CTRL+Q. Keep in mind that quiting is not elegant, if the user has not disconnected from the robot first.

About

This option displays a thank you message and the contact information of the developers, should you need any further information concerning KME . Its shortcut is CTRL+H.

A.3 Connection

Along with KME , you should have downloaded the python server (currently available only for the Nao robot), which serves as a mediator between the robot and KME . This program is interconnected using TCP/IP sockets, simultaneously with KME and the robot. Before using KME , one should run this python script. This is going to set up the connection between the server and the robot and afterwards will inform you about the port where it is waiting for a connection from KME . Then, you can run KME , if you haven't done so already, and you should enter that port number at the input box on the top left side of KME , and press the button "Connect". Well done ... You are now connected with the robot and you can start working with KME .

A.4 Joint Sliders

All Joint Sliders lie in a dedicated area on the right side of the GUI window. Each joint chain, is intentionally colored in the same color in order to help the user visualize the robot's joints easier. Furthermore, each slider is accompanied by an input area where you can directly enter and set the desired angle of the specific joint. The values, range, name and color of the slider are all set by determined

XML configuration file as discussed later in Section C. In general, you can choose your favorite way of moving slider, however when it comes to large changes in the value we suggest that you set directly the value as a float input, or click directly at the area of the slider you want to reach. Afterwards, by sliding left or right you can fine tune the angle. Please note that all values are represented in radians.

A.5 Browser area

In the lower area, there is a pose browser which holds the pose sequence of the motion we have developed so far. Each line is a pose represented as a series of values (one for each joint/slider). Depending on the platform a line may contain a piece of text accompanied by some specific parameters; for example KME supports talk commands between poses for the Aldebaran Nao robot. At any time you can highlight, one or more lines and execute any valid action from those available as buttons. These actions are presented below.

A.6 Push Buttons

A.6.1 Connect

This button initiates a connection to the IP address and port specified by the user. The input must be given in the following format: `IP:Port`. For example, in the frequent case that the user runs both the server and KME from a local machine, the input should be:

```
127.0.0.1:50000
```

The port given should match the one reported by the server (this is where the server is waiting for a connection). If there is a problem in setting up the communication, the user will be prompted about the exact problem with a message on a new window. After pressed, the "Connect" button will be deactivated and the "Disconnect" button will be activated.

A.6.2 Disconnect

In order to maintain port usability and minimize socket failures, every time that the user wants to finish a session with a real or a simulated robot, she has to press the "Disconnect" button to cleanly close the communication between KME and the server. Of course, in order to avoid the annoying situation of having a port not released by the Operating System, we dynamically assign the port number through the server script, starting at 50000 and increasing, until a predefined threshold, which is practically never reached.

A.6.3 Stiffness Radio Button

Having the ability to set the stiffness of the robot on and off, was a key factor in the development of KME , since most motion designing is based on hand-crafted motions, and further tuning by direct slider use. A value counter is used in conjunction with the stiffness radio button, to allow for the user to set the joints' stiffness at a percentage level, having 1.0 as the upper limit representing 100% joint torque. Try working with stiffness values lower than 1.0 in order to avoid high joint temperatures and abrupt movements which may cause damage to the robot. Please, note that if you have set manually the stiffness of the robot (outside KME), the stiffness value on the GUI will not be updated.

A.7 Motion Design

Motion Design is the main purpose of the existence of KME , thus through our needs and our experience we have managed to develop the following features that were in much need by the *Kouretes* team, while developing complex motion patterns.

A.7.1 Go To Pose

The "Go To Pose" button enables the user to send a specific pose highlighted in the browser to the robot. The robot will move to that pose (assuming stiffness is enabled) and at the same time the sliders will be set to the joint values of

the current configuration. Note that these values might not be exactly the same as the ones specified in the selected pose in the browser, since they are loaded directly from the current configuration of the robot, which may differ slightly due to obstructed movement, excessive stress, or low stiffness on some joints. By copying the real configuration to the slider, we protect the joints of overheating; they would be under constant stress otherwise. The time value however is taken straight out of the selected pose.

A.7.2 Update Pose

The "Update Pose" button allows the user to choose a line from the browser and update the values in that line with the new values specified by the current position of the sliders. This feature is extremely useful when only a few joint values in an existing pose need to be modified.

A.7.3 Capture Pose

The "Capture Pose" button corresponds to one of the most useful functions, since it asks the robot to report its current pose (all joint values). By capturing the robot's joint values, the sliders are automatically set to those values. Subsequently, if we want to save that particular pose, we have to click on the "Store Pose" button.

A.7.4 Store Pose

The "Store Pose" button saves the current configuration of the sliders as a new pose/line at the end of the browser list. Please note that the slider values represent real joint values only after capturing a pose from the robot.

A.7.5 Insert Pose

The "Insert Pose" button saves the current configuration, similarly to the "Store Pose" button, with the difference that the new pose is not inserted at the end of the browser list, but right after the currently highlighted pose. If no pose is currently highlighted, then the new pose is inserted at the end of the list.

A.7.6 Move Up

The "Move Up" button moves the highlighted pose one line up from its present position, essentially swapping it with the pose above.

A.7.7 Move Down

The "Move Down" button moves the highlighted pose one line down from its present position, essentially swapping it with the pose below.

A.7.8 Swap

In certain cases, we may want to swap the positions of two arbitrary poses, thus the "Swap" button swaps the positions of the first two highlighted poses.

A.7.9 Remove

The "Remove" button simply removes the highlighted entries of the browser. Please be careful, since this operation is irreversible, the removed poses are lost for good.

A.7.10 Check Buttons

A small number of check buttons is available to provide more usability for motion design. Through the check buttons, positioned between the two columns of sliders, the user can set the chain stiffness on and off to the left or right Arm Chain, left or right Leg Chain, and finally one option we found very valueable was to have access to the stiffness of both ankle joint on the two legs. This option is extremely useful, when creating motions by hand while holding the robot in the air and the ankle joints end up in random positions. In such cases, after finishing the design of the basic motion, for each pose in the sequence we can turn on stiffness on the entire robot body except for the ankle joints, place the robot on the ground, freely move its ankles to angles where full contact with the ground and balancing are achieved, and capture the resulting values.

A.8 Motion Execution

Motion Design is useless without the ability to test on the fly the motion designed so far. KME offers two options for motion execution and these are step-by-step execution and continuous execution of the whole motion.

A.8.1 Step Motion

If "Step Motion" is clicked, KME will start from the first highlighted entry in the browser and will set the joints of the robot to that pose (assuming a non-zero stiffness value). Clicking on "Step Motion" again, the next pose will be acquired, and the same pattern repeats as long as we click on "Step Motion" and the end of the list has not been reached. **Please, be careful that after a click on "Step Motion" the robot will linearly interpolate to the target pose from whatever pose it currently is. Please, be sure that the target pose is feasible from the current robot pose and always hold the robot while testing a motion at least for the first time.**

A.8.2 Play Motion

After testing the designed motion in a step-by-step manner, you may want to try to execute the complete motion. All precautions mentioned above are also applicable here. When it comes to robotics with physical moving parts you should always be cautious, not only to avoid damaging your robot, but also causing personal injury!

A.8.3 Time

Each line in the browser holds a time value in addition to the joint values. This value dtermines the transition time from the current pose to the next one in the sequence. The transition time can be modified through the box on the right side; clicking on the simple arrows increases/decreases the time by a tenth of the second each time; clicking on the fast forwad/backward arrows changes the time by one second.

A.8.4 Total Time

The total motion time is equal to the sum of all transition times in the sequence and is shown under the tabular area. Every time the pose sequence is updated in the browser, the total time is automatically updated to reflect its true value. The total time information is very useful when the designer has to meet tight duration constraints for the desired motion.

A.9 Time Scaling

Many times we find out that a motion we have designed should be executed faster or slower as a whole. Even though, we believe that this option is very important, we also realized that not many motion editing tools share this functionality. Thus, KME provides the means for scaling the total time of a motion through a scaling factor which is set to 1.0 by default. This factor multiplies all transition times. Setting it to a value higher than 1.0 will make the motion slower; a value less than 1.0 will make the motion faster. Therefore, the designer is able to test the same motion in faster and slower speeds and decide which one is best. It should be noted that KME calculates the scaled transition times dynamically, therefore the original motion is never overwritten.

A.10 Coupled Joints

In robots, as in nature, there seem to exist several symmetries. In order to take advantage of this property, we have considered the obvious symmetry on every humanoid according to the sagittal plane. For example, the Left Shoulder Pitch and the Right Shoulder Pitch are symmetric joints. Thus, we have created two drop-down menus for each pair of symmetric joints. The first drop-down menu determines the type of coupling and the available choices are :

Independent Each joint moves independently of its symmetric one.

Coupled The two joints must follow symmetric trajectories.

Mirrored The two joints must follow mirrored trajectories.

It is important to understand that when talking about a symmetric or a mirrored trajectory, we are not simply talking about giving the same values to both joints, or making one be the negative of the other. This is not applicable neither to the Aldebaran Nao robot, nor to other robots. To achieve successful coupling, KME uses percentages over the entire range of each joint to determine the corresponding symmetric or mirrored angle of a given joint angle. Thus, by taking the percentage within its range to where one joint has moved with respect to its minimum value, we can compute the corresponding angle of the coupled joint as the percentile offset within its range from its minimum value or maximum value depending on the type of coupling.

The second drop-down menu, allows the user to select the dominant joint in a pair of coupled joints. You can choose the right or the left as dominant and the other (the slave joint) will respond only to changes of the dominant joint. This feature may be thought as redundant as there is no need to have a dominant joint when two joints are coupled. This feature is in fact useful only at the initiation of a coupling session: when two joints are independent, their values are likely different; once they become coupled, one of the two joints must move to the coupled position of the other, but which one? Instead of hardwiring a fixed choice, we defer the freedom of choice to the designer.

Appendix B

Developing A New KME Server

The current version of KME provides a Graphical User Interface and a server application to work with the Aldebaran Nao robot. Extending KME to other robots, requires some familiarity with the communication protocol between KME and the server, before building a new server for another robot. The following pages explain in detail the entire procedure of developing a new server. Should you need any further information or help in developing your own server, feel free to contact me at `gpierris[at]gmail.com`.

B.1 Basic Concept

First, you should be familiar with the concept of the server. The server is an intermediate program, whose functionality is limited to data forwarding from the robot to KME and vice versa. A datagram of the whole concept can be seen in Figure 5.2. Before starting keep in mind that the server, whenever receives a message, should also send back to KME a message containing anything you like (alive is good choice, though). This is needed to ensure that KME can realize at any time if there is a connection loss due to a problem. In the newer version, we will look for a more elegant way of check this, but for now it works great.

B.1.1 Connections

Obviously, the server should be written as a TCP/IP server. It first waits for a connection from the robot and then from KME . The server should inform the user each time about the port number waiting a connection. In the provided Python server for Nao, the server-robot connection has been replaced by a very handy and more elegant approach, that of setting up one proxy for every functionality. In our case, for the Aldebaran Nao robot, we set up three proxies; one motion proxy, one speech proxy, and one proxy for other information, such as the battery state. The connection between the server and the robot is trivial and is up to you to choose how to implement it. After some search, I am pretty sure that you will find an example for your robot, and even if you don't, the bibliography is full of examples for every programing language you choose to develop your server application.

Afterwards you should open a new server socket, and inform the user about the available port. In order to iterate through ports and find the available one, you can use the following example in Python.

```
for i in range(10):
    try:
        kmeSocket.bind ( ( '', editorPort ) )
        break
    except Exception:
        editorPort += 1

print( "Waiting connection at Port :" + str( editorPort ) )
```

This piece of code in every iteration, tries to bind the a port. If that fails, increases the port number by one until a port becomes available. From our experience in the second or third time, or in the worst case that we have used KME two or three times without disconnecting the clean way, we could bind the port. Thus, we think ten times is an acceptable limit. At this point, the server simply waits for a connection from a KME client.

B.2 Main Routine

The server performs a `while(true)` loop, listening to the socket for incoming messages from KME . The decision to use a blocking function, such as `recv`, or a non-blocking `select` is up to you. The difference lies in the fact that, when you use `select`, you can have multiple KME clients connected to one robot, a choice that is highly discouraged. One other interesting application could be to have only one server, for example in the laboratory, which serves as the mediator for every distinct KME - robot connection. In general, you have to freedom to develop the server in every way you want.

Please, note that message decoding can be told from the first characters or the first word in an ASCII message. In case you just want to find a word or a pattern inside the message, note that first you should decode the `talk` message, if you are working with the Aldebaran Nao robot, before anything else is done. I did not do that in one of the very early versions of the server, and once I sent the robot the string `stiffness` to use the TextToSpeech module, and the server crashed. The string included the pattern `stif`, thus the server was waiting some extra parameters to define the value of the desired stiffness. Instead of these parameters a lot of other information regarding the speech module were passed crashing the server. This was just a note to be carefull with the `switch-case`, or `if` structure you are going to use during decoding.

B.2.1 Messages Received

Once the connection has been established, we are ready to exchange messages between the server and KME . The types of messages sent by KME are ASCII characters in order to be easily manipulated and understood by the user. Please, adhere to this convention in order to avoid wasting time for debugging; KME will not understand the messages and the result is unpredictable.

B.2.2 Motion Messages

The motion messages are defined by the header `play` or `pose`. These messages are exactly the same, but starting with another word, helps the user tell the difference

of a pose sent by the sliders and another sent through the "Step Motion" or "Play Motion" buttons. What corresponds to what is obvious! The rest of the message is formed by listing the slider values in a series that is well understood by both the server and KME ; different slider values are separated using a percentage sign (%). At the end of the string, we concatenate the time variable and the word `end`, again separated with the percentage sign (%). This is used because the number of joints is dynamically set by the `xml` configuration file and therefore may vary. An example of a message can be seen below:

```
play%-0.023%0.048%1.173%0.446%-0.591%-0.538%-0.201%-0.054%...
-0.503%1.331%-0.787%0.025%0.000%0.011%-0.537%1.385%-0.830%-0.021%...
1.261%-0.362%0.666%0.552%1.200%end
```

The above example is valid in the Aldebaran Nao robot and the sequence of the servos follow the sequence below:

```
HeadYaw, HeadPitch, LShoulderPitch, LShoulderRoll, LElbowYaw,
LElbowRoll, LHipYawPitch, LHipRoll, LHipPitch, LKneePitch,
LAnklePitch, LAnkleRoll, RHipYawPitch, RHipRoll, RHipPitch,
RKneePitch, RAnklePitch, RAnkleRoll, RShoulderPitch, RShoulderRoll,
RElbowYaw, RElbowRoll
```

B.2.3 Load Message

When the user presses the "Capture Pose" Button, the server receives the string `load`. The server as a response must send a string containing the values of all the joints of the robot including % characters between consecutive values. So, one possible example for a robot could be:

```
0.234%1.534%0.142%...%1.123
```

What we found to be quite interesting were the two possible ways (perhaps there exist more?) of getting the values of the robot joints from two different version of the Nao robot. The first way used in the Aldebaran Nao V3 robot is straightforward: we make a call to the `vector<float> getBodyAngles()` function in the API and just wait for the robot to give us the correct values. The other way used in the Aldebaran Nao V1 robot was not that clear. The API provided two functions: `vector<float> getBodyAngles()` and `vector<float> getBodyAngleErrors()`. The first function was returning a vector of the last user-assigned joint values. Subsequently, when we were turning the stiffness off, we were receiving different values than the real ones, using the `getBodyAngles()` function. Additionally, the `getBodyAngleErrors()` was returning a vector with the error values from the last user-assigned values, thus we came up with the solution of adding those two vectors. The error is the distance between the last user-assigned joint values to the current (true) joint values, so adding the error to that last command returns the real (true) values. In any case, please consult your robot manual to distinguish and clear up every issue concerning its motion features, to ensure your choices every time.

B.2.4 Stiffness Message

For the stiffness message, KME simply sends `stif_on` and the corresponding value of stiffness between 0.0 and 1.0. To set the stiffness off, KME simply sends `stif_off`. Thus the messages can be of two types:

`stif_on%0.9`

or

`stif_off`

For the Aldebaran Nao robot, some more joint-specific messages may be exchanged. These include, setting the stiffness in a chain or in each ankle separately. These messages use the same general pattern and include some additional headers.

`x_y_stif_z`

The server should decode this message in the following way. `x` can either be `l` or `r` representing left and right respectively, `y` can be `arm`, `ankle`, or `leg`, and `z` can be `on` or `off`. When receiving such messages, the server should set the desired stiffness to the desired joint chain. An example of such message could be:

```
l_arm_stif_on\%0.9
```

which will set the left arm's stiffness to 90% of the maximum stiffness. Another example could be:

```
r_leg_stif_off
```

which will set the stiffness off to the right leg of the Nao.

B.2.5 Disconnect

In order the ensure clean closing of the sockets, KME sends a `disconnect` message to the server. After receiving this message, the server should cleanly close the sockets.

B.2.6 Speech Messages

Speech messages are only available for the Aldebaran Nao robot. From the left tabular area of KME , the user can have extended access to the `ALTextToSpeech` module, in order to allow for the user to create basic behaviors. The talk message is quite long and contains both the text and some extra parameters. An example of such a message would be :

```
talk\%This is a test\%Volume\%50.00\%doubleVoiceLevel\%x.xx\%
doubleVoice\%x.xx\%pitchShift\%x.xx\%doubleVoiceTimeShift\%x.xx\%
Voice\%x\%SaveToFile\%x
```

The server receiving this string is going to send the string `This is a test` to the robot and will also set all the parameters to the corresponding values provided. More information on how to set the speech parameters can be found in the NaoQi documentation.

B.2.7 Robot State Messages

One option that is very useful during the development of a motion is the visualization of the battery state and other platform specific information, such as the temperature of the joints. Currently, we are only using the battery state and we visualize it on KME . KME sends a `getRobotState` message, and the server, using the ALSentinel proxy, replies with a string containing one of the five possible numbers returned by the function `getBatteryLevel()`. We intend to include more robot state information in the future.

Appendix C

KME XML Configuration Files

Kouretes Motion Editor from its conception was aiming for the Aldebaran Nao robot. In order to make KME able to support other robots and available to other researchers, we developed an XML parser, in order to deliver a customized KME for every robotic platform we intend to work on. This feature is very important for robotic researchers, since working on different platforms is now a piece of cake.

Creating correct XML files is of high importance to make KME work correctly. If you do not have the XML file provided for the Aldebaran Nao robot, download it from www.kouretes.gr. If you are creating your own configuration file for your robot, please consult these instructions. If you do not manage to create an appropriate configuration file, feel free to contact me.

The current format of the XML file includes platform-specific information about the robot, its joints, the color of the sliders, and the order in which the user wants to arrange the values of every joint when send to the server. An excerpt from the configuration file for the Aldebaran Nao robot is shown below.

```
<Robot>
<manufacturer>Aldebaran-Robotics</manufacturer>
<type>Nao</type>
<JointNumber>22</JointNumber>

<Joint>
<name>HeadYaw</name>
```

```
<minBound>-120.0</minBound>
<maxBound>120.0</maxBound>
<step>0.0</step>
<color>24</color>
<coupledWith>None</coupledWith>
<couplingType>None</couplingType>
<indexOrder>0</indexOrder>
</Joint>
.
.
.
</Robot>
```

There are some rules and restrictions you must comply with to generate correct configuration files. Please, note that everything mentioned in this section is compulsory, since otherwise KME will not be able to run. Our short-term plan includes building an XML parser and a validator to ensure that your XML is fully compliant with KME .

C.1 XML Configuration File

C.1.1 Robot Information

First, you must start off with the element `Robot`. This is the beginning of the XML file. Subsequently, you must continue on setting the robot manufacturer, the type of the robot, and the number of the available joints. For example, in the Aldebaran Nao robot, the foresaid elements are `Aldebaran-Robotics`, `Nao`, and `22`.

C.1.2 Joint

Right after the general information, the user must enter one `Joint` element for each joint of the robot. The sequence you enter these joints will determine their order in the KME window. The first half of the joints are going to be placed in

the left column of the sliders, while the rest are going to the right column. So, we provide the user the freedom, to place the joints in any order she wants. For example, if one is working only with the robot legs, why not have the ability to place those sliders on top of all others?

C.1.3 Name

The `Name` element is the string appearing as the name of the slider in the KME window. You can choose whaever name you prefer for each joint to help you identifying the corresponding slider in the graphical user interface.

C.1.4 Min- and Max- Bounds

The two next elements are `minBound` and `maxBound`. These are used to set the bounds of each joint. KME will never send to the robot an angle outside these limits. These limits must be in degrees or rads or whatever you like. This step is important especially in robots that do not check angle values before sending them to the joint controllers.

C.1.5 Step

The `step` element is used more often in cases where joints cannot take advantage of the whole range of the joint, and less often when the user does not want to have fine precision over the joint range. Setting the step to `0` means that the slider will be freely able to move anywhere inside the range of the slider (continuous values). If the step is larger than zero, then the slider will take discrete values and the stepsize will be determined by the step, allowing the slider to move only in multiples of the step each time. If you have a binary joint with only two possible values, then you must set the `step` element to `-1`, and the bounds must be set to those two values.

C.1.6 Coupling

The coupling feature is the distinctive characteristic that differentiates KME from any other motion design tool. The value of `coupledWith` element can be either

None or Coupled. The coupledWith element allows you to couple one joint with another one. By coupling, we mean that these two sliders are grouped and they have the ability to become coupled if needed through the menus of KME . Please, be aware that you must use valid joint names for KME to run flawlessly. You should keep in mind that when you declare, for example JointA to be coupled with JointB, you do not have to state again in the JointB element that it is coupled with JointA, but we suggest you do. Actually, you can avoid writing it again in JointB, but we do so because KME gives you another innovative feature. This is the ability to couple joints that do not share the same characteristics. For example, coupling LShoulderPitch with RShoulderPitch is obvious, but coupling LShoulderPitch with HeadYaw, which is in turn coupled with LHipRoll which is coupled with RHipPitch is not obvious and really exciting at the same time. For now, we allow this feature, but there might exist graphical issues with the placement of the drop-down menus. In future versions this will be fixed.

C.1.7 Coupling Type

The couplingType can be of three types: None, coupled, or mirrored. Using coupled, we mean that two joints coupled with each other will assume symmetrical positions with respect to each other. When mirrored is used, the slave joint follows the dominant in an inversely proportional way. For example, when the LShoulderRoll is mirrored with RShoulderRoll, then if the first has its arm closely attached to the body, the other is wide open.

References

[1] D. Gouaillier and P. Blazevic, "A mechatronic platform, the Aldebaran robotics humanoid robot," *32nd IEEE Annual Conference on Industrial Electronics, IECON 2006*, pp. 4049–4053, November 2006. 1

[2] H. Kitano, M. Asada, Y. Kuniyoshi, I. Noda, E. Osawa, , and H. Matsubara, "Robocup: A challenge problem for AI," *AI Magazine*, vol. 18, no. 1, pp. 73–85, 1997. 5

[3] G. Pierris and M. Lagoudakis, "An interactive tool for designing complex robot motion patterns," *IEEE International Conference on Robotics and Automation (ICRA), Kobe, Japan*, May 2009. 29

[4] B. Spitzak, M. Sweet, C. P. Earls, and M. Melcher, *The Fast Light Toolkit v. 1.3 Programming Manual*. [Online]. Available: www.fltk.org 31

[5] O. Michel, "Webots: Professional mobile robot simulation," *Journal of Advanced Robotics Systems*, vol. 1, no. 1, pp. 39–42, 2004. 41

[6] Aldebaran Robotics, *Choregraphe*. [Online]. Available: www.aldebaran-robotics.com/eng/choregraphe.php 53

[7] Microsoft Hellhounds, *Team Report 2006*, Dortmund University, 2006. 54

[8] Sony Corporation, *MEdit for ERS-7 1.0.7, OPEN-R SDK*, 2003. 54

[9] *Skitter v3.40*. [Online]. Available: www.dogsbodynet.com/skitter.html 55