ICIT 2015 The 7th International Conference on Information Technology
doi:10.15849/icit.2015.0012

Experiments with Simulated Humanoid Robots

Hans-Dieter Burkhard
Humboldt University Berlin
Institute of Informatics
Berlin, Germany
nao-team@informatik.hu-berlin.de

***Abstract*— Experimenting with real robots is limited by the available resources: Complex hardware is costly, and it needs time and experience for setup and maintenance. Simulated robots can be used as alternative. Our RoboNewbie project is a basic framework for experimenting with simulated robots. It serves as an inspiration for beginners, and it provides room for many challenging experiments. The RoboNewbie agents run in the simulation environment of SimSpark RCSS, the official RoboCup 3D simulator, where the simulated robots are models of the humanoid Robot NAO of the French Company Aldebaran. Different example agents provide easily understandable interfaces to simulated sensors and effectors of the robot as well as simple control structures. The framework has been successfully used at different courses where the participants needed only few hours to understand the usage of the framework and to develop own agents for different tasks.**

***Keywords*— Robotics, e-Learning, Simulated Robots, RoboCup**

I. INTRODUCTION

Understanding grows with active commitment: to "do" something, to master it, provides a deeper understanding. Experiencing with own experiments is an important prerequisite for studies in Robotics and Artificial Intelligence. But experimenting with real robots is difficult not only because of expensive hardware. Maintaining the robots and set ups for experiments are very time consuming even for experienced people. Experiments at home as needed for e-learning would require a deep technical understanding by the students, i.e. experiences that they are just going to learn. So it is not surprising that simple hardware is still broadly used in robot experiments, hardware which is far behind the recent technical developments, not to talk about e.g. complex humanoid robots. The collection of papers [1] can be understood as an illustration of that statement.

Simulated robots in simulated environments are widely used as an alternative for complex hardware. They are often simulations of existing robots and serve for preliminary programming, tests and evaluations. Because of the "reality gap", the transfer of programs from simulated to real robots is a non-trivial task [2]. Reducing the reality gap needs increasing efforts in the simulation and leads again to complex systems which need more efforts by the user. The trade-off must be handled carefully for systems better suited for beginners.

The RoboCup community has more than 15 years of experiences with real and simulated robots in the field of soccer playing robots [3]. Soccer playing robots have been established as a challenging test field for the progress in scientific research and technical developments. Robots have to be able to control their bodies and their motions according to soccer play, they must perceive a dynamically changing environment and they have to choose successful actions out of many options in real time. They have to cooperate with team mates and to pay attention to opponents. Several thousand scientists and students are participating in the annual RoboCup competitions in different leagues with different types of real and simulated robots. The humanoid robot Nao of the French Company Aldebaran [4] is used in the Standard Platform League, while its simulated version is used in the 3D-Simulation League. The official SimSpark RoboCup 3D Soccer Simulation (SimSpark RCSS) [5] provides an excellent environment for experiments with simulated complex robots (see Section III). It provides a physical simulation using ODE [6] for the body dynamics of the robot Nao and the soccer environment.

Our RoboNewbie Project is a basic framework based on JAVA for the development of simulated humanoid robots. It provides easily understandable interfaces to simulated sensors and effectors of the robot as well as a simple control structure. It runs in the environment of the SimSpark RCSS, thus it can but need not be used for soccer playing robots. Users can develop their own motions, e.g. for dancing, gymnastics or kicking a ball.

The RoboNewbie Project implements some kind of "minimalistic approach" with respect to Robotics. Users are able to start without special knowledge about robots. They can learn by their own experiences about the basic concepts of perception, motion, control, synchronization, and integration. All related program code in RoboNewbie is understandable from simple principles without further knowledge. That concerns the structure of the code as well as the underlying computational methods. As soon as users learn more about Robotics, they will be able to extend the programs accordingly, e.g. concerning complex motions or world modeling.

doi:10.15849/icit.2015.0012

Moreover, the framework has also good potentials for the research on foundations. e.g., on computational models as well as on different problems in cognitive science. It can be useful in verifying models and in gathering large data sets for experiments in data mining.

The paper is organized as follows: After an overview about the concept and the downloadable resources of the RoboNewbie project, it gives a short overview about SimSpark RCSS. The communication between the RoboNewbie agents and SimSpark RCSS is described next. The main part of the paper in Section V discusses the details of the RoboNewbie framework, and the paper ends with results of practical evaluations and conclusions.

II. The RoboNewbie Project and its resources

The main goal of the RoboNewbie Project is to provide an uncomplicated starting point to the programming of complex robots with minimal requirements and pre-knowledge. The users are only supposed to have some programming background (Java) and some technical/mathematical understanding. More knowledge about Robotics can be provided in parallel to the exercises with RoboNewbie, e.g. in introductory tutorials (as was already done) or by e-Learning material.

The objective behind RoboNewbie is the realization of the following requirements:

- Holistic view on robots: For beginners, it is more appealing to see a robot behave like a human than to test and calibrate the behavior of a sensor. Of course, when dealing with more complex tasks, users will experience the need to have better knowledge about the usage of sensors and actuators, and then they may draw their own conclusions.
- Motivating scenario: Application fields from daily life with known properties and rules are well suited. Robots which imitate human skills are especially motivating.
- Scalable tasks: Inexperienced users should have no difficulties to perform first steps with own experiments and later move to more complex tasks with unlimited challenges.
- Low requirements: The usability would be restricted if people need pre-knowledge on Robotics or if they are supposed to have deep knowledge in hardware and software. Basic programming skills and interests in mathematics and natural sciences should be sufficient.
- Low costs: The costs of a learning system include money and efforts for purchase, set up, and maintenance, respectively. They should be as low as possible to permit a broad usage.

The users of the RoboNewbie project can find all materials on the web page of Berlin United -- Nao Team Humboldt [7].

Besides links to RoboCup, Nao (Aldebaran) and the SimSpark-Wiki, it contains resources for download:

- Description of installation and first steps.
- Sources of RoboNewbie agents programmed in JAVA 7 and prepared for usage under Netbeans.
- "Quick start tutorial": Introduction to the features and the usage of the agent.
- Motion Editor for the design of Keyframe Motions (needs JAVA 3D to be installed).
- SimSpark RoboCup 3D Soccer Simulation (SimSpark RCSS) for Windows with an introduction to SimSpark RCSS as far as needed for RoboNewbie.

All provided code is open source. Some parts of the RoboNewbie code use code of the RoboCup team magmaOffenburg [8].

Figure 1: SimSpark RoboCup 3D Soccer Simulation as used in the RoboCup competitions. The field size is reduced for RoboNewbie.

III. SimSpark RoboCup 3D Soccer Simulation

SimSpark RCSS is developed and used by the RoboCup community in the 3D simulation league. SimSpark is a generic physical multi agent simulator system for agents in three-dimensional environments. It uses the Open Dynamics Engine (ODE [6]) for detecting collisions and for simulating rigid body dynamics. ODE allows accurate simulation of the physical properties of objects such as velocity, inertia and friction.

doi:10.15849/icit.2015.0012

The Simulator SimSpark RCSS consists of two programs (server for simulation and monitor for visualization and interaction) together with configuration files. It models a soccer field with the player bodies (adapted from the robot hardware of Nao) and the ball. It also controls the rules of the soccer game, i.e. it controls the game according to the decisions of a simulated referee.

SimSpark RCSS can be used as open source software. This was also an important criteria for its usage. It can be downloaded from [5] for different platforms. A complete preconfigured version for Windows 7 is provided for RoboNewbie which can be downloaded from the RoboNewbie web page [7]. Nevertheless, the RoboNewbie agents run with SimSpark RCSS under other platforms, too.

By some small changes in the configuration files, the soccer rules are simplified for first usages with RoboNewbie. The SimSpark RCSS project itself is constantly evolving according to the progress in the RoboCup initiative. The version (compiled in June 2012) on the RoboNewbie web pages serves for stable usage and avoids potential incompatibility problems by new RoboCup versions.

SimSpark RCSS is documented in a Wiki [5] with download links to the latest versions as used in the competitions. The Wiki documentation is thought to represent the actual state of the simulator by continuous updates. But since different developers are volunteering in parallel on different tasks in the project, the structure of the Wiki is not always optimal, and occasionally some outdated information is still present. Moreover, the Wiki is directed to experienced users. This makes it sometimes difficult to understand for novices. According to our experiences (cf. Section VI), the deeper knowledge is not needed by beginners.

To provide an easy access, the downloads of the RoboNewbie project contain an introduction to SimSpark RCSS which refers to the provided version (as described above). It gives the user an overview about

- Simulation using SimSpark RCSS: The SoccerServer and the Monitor.
- The Nao-Model used by SimSpark RCSS.
- Communication between Agents and SimSpark RCSS (with explanations of the message formats as background information).
- \Synchronization between SimSpark RCSS and the Agents.
- Monitor and User Interface.
- Running a Game.

Actually, our description of SimSpark RCSS provides also some "background" information which is not needed for beginners, e.g. details about the message formats. Since RoboNewbie permits an easy and direct access to the items of messages like sensor values and motor commands, the syntax of messages must not be known by users. Nevertheless, we have included the information for deeper understanding of RoboNewbie in case of interest.

IV. Communication between Agents and SimSpark

SimSpark RCSS implements the soccer environment including the bodies of the Nao robots. It models all physical interactions between players, ball and environment. The agents implement the control of the players.

The interface between the physical environment and the control of real robots is constituted by sensors and actuators: Robots perceive the world by sensory data (e.g. by vision, accelerometer, force sensors etc.), and influence the world by their actuators (motors, voice etc.).

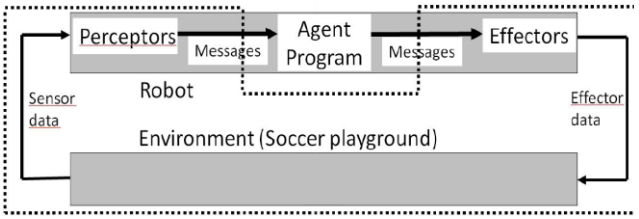

Figure 2: Simulation scheme with message transfer between the agent and the simulated physical world. The simulated physical world consists of the robot hardware and the environment. The agent controls the actions of the robot. Messages contain data of the preceptors (sensors) and effectors (actuators).

In simulation (Figure 2), the sensory data are calculated by the simulator according to the situation in the simulated world (e.g. observable objects) and sent via messages to the agent. Then, like a real robot, the agent can update its belief about the situation and decide for actions it wants to perform. A real robot would then activate its actuators (e.g. motors at the joints) to perform the intended actions. In simulation, the agent communicates with SimSpark RCSS again by messages which transmit the effector commands. Both are synchronized by a communication cycle of 20 milliseconds.

In SimSpark RCSS, the message transfer is optimized for minimizing the server and the traffic load: All sensory data are packed in one server message to be sent at the beginning of a communication cycle. Vice versa, the agent can send all action commands by a single agent message before the end of a cycle. This trick makes it possible to run several agents together with the simulator even on a simple laptop.

The message formats follow a special syntactic scheme based on symbolic expressions (S-expressions). As a consequence of collecting data into one message, the preparation of the data in an agent needs more efforts than in a real robot. It is a special feature of the RoboNewbie agents that this preparation is hidden from the user: The agents provide special getter- and setter-methods which allow the access to the perceptor (sensor) data and the setting of effector (actuator) commands in a similar way as in a real robot.

doi:10.15849/icit.2015.0012

The interaction between the server and the agents works as follows (see Figure 3):

1. At the beginning of a cycle at a time t, the server sends individual server messages with sensations to the agents.
2. During this cycle, the agents can decide for new actions depending on their beliefs about the situation.
3. Before the end of this cycle, the agents should send their agent messages to the server for desired actions.
4. The server collects the messages of all agents and calculates the resulting new situation (poses and locations of the players, ball movement etc.) according to the laws of physics and the rules of the game. This is done during the following cycle at time t+1, i.e., the server message sent at the beginning of this cycle regards the situation calculated in the previous cycle at time t. We have a reaction delay as in reality (see Figure 3).
5. At the beginning of the subsequent cycle, at time t+2, the sensor data in the server message is based on the effects of the actions at time t+1 which were chosen by the agent according the information from time t.

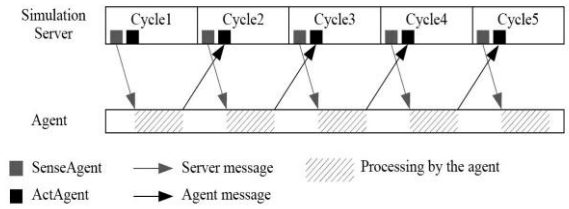

Figure 3: Message exchange during the server cycles.

A special feature of SimSpark RCSS is the use of so-called perceptors instead of sensors. The perceptor data can be regarded as already pre-processed sensor data. For example, the image data from the camera are not presented by a pixel matrix. Instead, the vision perceptor sends a collection of observable objects with egocentric coordinates relatively to the camera of the observing agent. In a similar way, action commands of the agent are encoded as so-called effector values and sent to the server which translates them to motor control commands. The calculation of perceptor values and the interpretation of effector values are part of the simulator, too.

On the agent side, a server message has to be parsed for the contained perceptor values, and the action commands have to be collected to the agent message. Both constitute a significant burden for a beginner while it provides only few insights to robotics. The RoboNewbie users need not to care about that, because the RoboNewbie agents do all this work in the background.

Besides some effectors related to initial connection with SimSpark RCSS, there are hinge joint effectors for each of the 22 hinge joints (see Figure 4) and a say perceptor (as of a loudspeaker with limited capacity). The following perceptors are available in SimSpark RCSS (for details see the Wiki or our SimSpark description):

- Vision Perceptor (as of a camera in the center of the head).
- Hinge Joint Perceptors at each of the 22 hinge joints.
- Accelerometer in the centre of the torso.
- GyroRate Perceptor in the centre of the torso.
- Force Resistance Perceptor at each foot.
- Hear Perceptor (as of a directed microphone with limited capacity).
- Game State Perceptor (reporting the actual game state of the soccer match).

V. RoboNewbie Framework

The RoboNewbie framework offers a comfortable interface for agents interacting with SimSpark RCSS. It includes sample agents which illustrate basic concepts and methods of Robotics and Artificial Intelligence. Users can start exercises with these agents and learn how to use RoboNewbie and what the programming of robots is like. They can make their own experiences with different topics and algorithms by modifications and extensions.

It was a main goal of the project, to provide easily understandable concepts, methods and programs. There are no complicated structures, and all code is documented in detail. As a consequence, some more demanding concepts were replaced by simpler approaches (e.g. keyframe motions instead of inverse kinematics, approximated coordinates of observed objects etc.). Nevertheless, the clear structure of the project supports extensions for more challenging solutions if wanted.

A. Low Level Interface Functionalities

The framework includes interface functionalities on two levels. The lower one corresponds to the hardware-near functionalities of robots, while the higher one is concerned with more abstract control functionalities. Especially for the lower level, parts of the code of the team magmaOffenburg [8] were used by us as documented in our source files.

The hardware-near layer encapsulates the network protocol for interaction with SimSpark RCSS and it allows access to the simulated hardware entities corresponding to sensors and motors. The access is implemented by getter functions for perceptor values of different perceptors which can be used similar to sensor signal queries of real robots. Related setter functions for effector values can be used for the control of actuators. Especially the low level interface functionalities for SimSpark RCSS are a hurdle for beginners and need time consuming work even for experienced users.

doi:10.15849/icit.2015.0012

They concern tasks like network connection, synchronisation with the server, parsing of nested server messages, syntactical analysis of S-expressions, synthesis of agent messages with a lot of technical non-robotics details. The users of RoboNewbie need not to care about all this details, the framework offers ergonomic methods for the interaction with the simulated environment in an easily understandable way similar to the methods used by the operating systems of real robots. Users can learn to use these methods after a short training time (cf. the evaluation in Section VI).

The synchronization protocol was already described in Section IV. The user needs not to care about the communication, except the delays by the protocol and the duration of the cycles given by 20 msec. It is necessary to fetch a server message at each cycle and to send the agent message before the end of the cycle. The related control structures are already implemented in the examples and explained by the tutorial. Hence, if the calculations during one cycle do not exceed the cycle time, there will be no problem. The time needed depends of course on the used computer. The example agents run without problems even on less powerful machines.

The first example "Agent_BasicStructure" in the tutorial let the users start with an agent which already implements all low level communication. The agent simply rises an arm by setting related effector values. The user can experiment with other values and other effectors just to understand the basic structures.

B. Perception

The available perceptors were already listet in Section IV. All perceptor values can be queried by related getter methods using the perceptor names instead of the acronyms of the server messages. This allows a comfortable access to the perceptor data which corresponds to the access of sensor values by a related operating system of a real robot.

The necessary conversions from the nested server messages to the perceptor values are already implemented in the RoboNewbie framework. For that, the server message are parsed for the constituents of a tree like structure (again, thanks to the code of the team magmaOffenburg [8]). According to the analyzed acronyms in the expressions of the tree, the corresponding perceptor values are filled in by RoboNewbie.

The programs "Agent_TestPerceptorInput" and "Agent_TestLocalFieldView" illustrate the usage of the related getter methods and the perceptor values. The examples serve also as an illustration to the usage of the logger functions described below in Subsection E.

As an exercise of the tutorial, the user can implement an agent, which lifts the robots arm, when it senses another robot and moves the arm down, when it does not sense any robot. Which arm is lifted should depend on the side where the other robot is seen.

Special efforts are needed for the vision perceptor. It provides coordinates of all objects in the vision range of the camera. SimSpark RCCS in its common version does not communicate image data. Instead, the communicated information can be understood as the result of basic image interpretation, it contains coordinates of the goal posts, the lines, the ball, and the body parts of robots.

The vision perceptor provides values by egocentric coordinates relatively to the camera in the centre of the head. Since the head may be turned and tilted, further calculations are necessary to get the coordinates of objects relatively to facing forwards. To get the coordinates relatively to the feets on the ground, it needs further calculations. Accurate calculations would need the inspection of the cinematic chain. The necessary data are available by the hinge joint perceptors. Further calculations including self localization are necessary for the transformation into global coordinates. RoboNewbie does not provide related programs following the intended "minimalistic" approach, because they would not be understandable by beginners without pre-knowledge about Robotics. But the implementation of related methods can serve as exercises during courses in Robotics.

As a simple substitute, we have decided to provide only approximations for the conversion from camera coordinates to facing forward coordinates. They are documented in the sources and easily to understand. Users can make experiments according to the accuracy and draw own conclusions on cinematic relations.

Visual information is provided by SimSpark RCSS only at each third cycle, and the robot would have to act blindly in between when there are no vision data available. Hence, the vision information should be stored for the following cycles. Moreover, the vision perceptor is limited by the camera view range of 120 degrees horizontally and vertically. The robot has to move its head to observe more objects in the world.

Again it is useful to store objects seen before in other directions. In general, such updating and memorizing of observations is maintained as belief of the robot in a so called world model. Updates may regard corrections according to robot motion, guesses for movements of invisible objects and integration of information communicated by other robots.

Again, a fully elaborated world model is far behind the scope of beginners. Hence, RoboNewbie provides a very simple version, where just the observed objects are stored in a simple form. The coordinates of those objects are referenced with respect to the facing forwards coordinates. Turnings of the head are already regarded by RoboNewbie, but only by the approximate calculations as described above. Other movements of the robot like turning or walking are not regarded. Time stamps indicate the last time of observing an object. The example "Agent_TestLocalFieldView" illustrates the perception features of RoboNewbie.

C. Motions

All intentional motions are performed by controlling the hinge joints (see Figure 4) by sending effector values (defining the speeds of motors) to SimSpark RCSS. Then the physics simulation engine calculates the effects of the commands regarding physical laws and updates the simulated world accordingly.

doi:10.15849/icit.2015.0012

Simple motions like turning the head or rising the arms can be easily programmed by the users following the already mentioned examples. The motions can be controlled using the feedback of hinge joint perceptors. i.e. by sensor-actor coupling, where the delay of observing an action has to be regarded as described in Section IV. There is much room for own experiments of users.

More complicated motions like walking need coordinated movements of different joints. Users may learn about these problems after some trials. We have decided to provide keyframe motions in RoboNewbie because they are easily to understand and to design. The interpolation mechanism for keyframe motions in RoboNewbie realizes a linear interpolation - users may implement other interpolation methods like splines if they want. Keyframes are stored as text files which can be edited by any text processing system. Users can even design and change motions while using the programs as a black box.

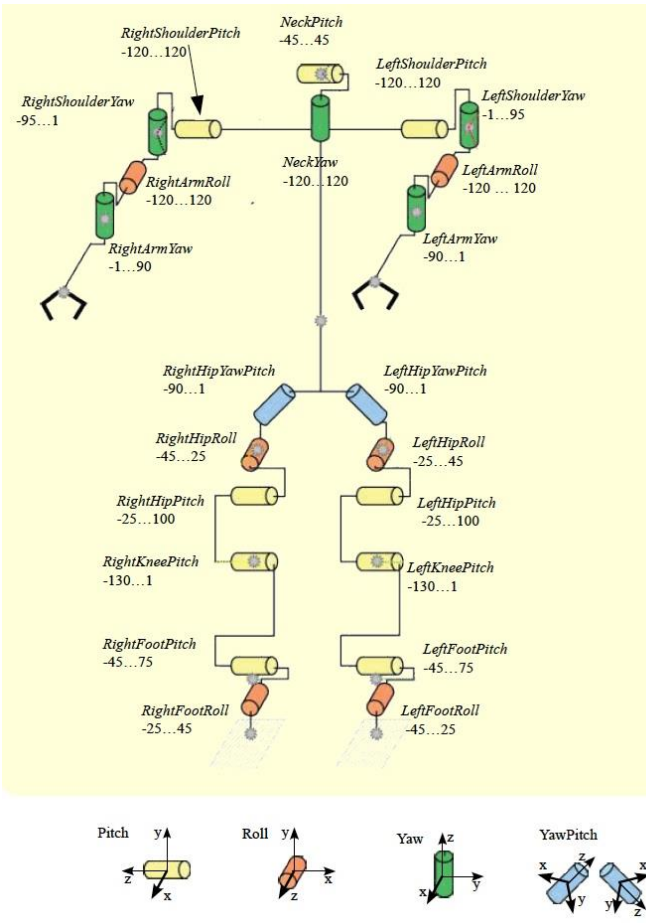

Figure 4: Joints of the robot. The range of the joint angles are given in degrees. Picture adapted from [5].

RoboNewbie contains a set of predefined keyframe motions for walking, turning, stand up and others. Users can change these motions (the related text files). New motions need an integration as explained in the tutorial and the documentation.

According to simplicity, there are no concepts implemented for interruption of motions: Each motion is performed completely until its end, and there are no cyclic motions, e.g. for walking. Instead, continuous walking can be performed by subsequent calls of a two-step-walk.

The design of keyframe motions is supported by a graphical Motion Editor. It can be downloaded from the RoboNewbie Web page as well. It shows the postures of the robot for selected keyframes. Then the keyframes can be edited in two ways. In the graphical representation the posture can be kneaded into the desired posture with the mouse. Alternatively, each joint angle can be set to specified values which are immediately presented by the graphics. Transitions between keyframes can be defined with specific transition times resulting in a keyframe sequence as usual.

The program "Agent_KeyframeDeveloper" helps designing keyframes. A robot performs the motion of the actually edited keyframe file. After each change, the new motion is performed immediately. If the robot falls down, it stands up by itself.

The example "Agent_SimpleWalkToBall" illustrates the motion concepts. As an exercise of the tutorial, the users can change that program to implement obstacle avoidance (walk around the ball without touching it). They can use motions for walk, stop and turn. Additionally, the agent must be able to recognize the ball and to decide for the appropriate motion according to the ball position. Another exercise is the design of a new motion for kicking the ball. Users can furthermore do their own experiments e.g. with dancing robots.

In general, keyframe motions are useful for special motions like standing up, but they are not so well suited e.g. for walking. Walking is still a challenging problem in Robotics. The users of RoboNewbie will get some understanding about the task. Moreover, the framework is well suited as a basis for other implementations and for Machine Learning by more educated users. But according to our "minimalistic" approach, related implementations are not provided.

D. Control Cycle and Decision Making

The basic control cycle follows the classical centralistic deliberation approach, often denoted as the "sense-think-act-cycle", or by similar names. This corresponds closely to the cycle given by SimSpark RCSS: At first, sensations are provided to the agent, then the agent decides for appropriate plans and then it sends the related action commands back to the server.

To realize concepts of Embodied Robotics/AI it would be necessary to have local sensor actor coupling, distributed control, embodiment, situatedness, emergent behaviour etc. The real robot Nao as well as its simulated counterpart with the central control (i.e. our agent) is not primarily designed for such purposes. It is possible in principle to design sensor actor couplings and other behavioral concepts in the RoboNewbie framework. One might even split the agent into different "parallel" acting parts (implemented e.g. by threads) to simulate

doi:10.15849/icit.2015.0012

distributed controls, but some synchronization is unavoidable by the server cycles of SimSpark RCSS.

At the same time, thinking in terms of the "sense-think-act-cycle" is quite natural for beginners because it reflects some causal dependencies. It provides an intuitive and easily maintainable structure in the design of robots. Therefore, the control cycle in RoboNewbie adopts the related terms for structuring the run-methods of the agents by cyclic calls of methods sense, think and act. The think-method is sometimes omitted in case of simpler ("reactive") agents.

The sense method is responsible for receiving and processing the perceptor data by the related RoboNewbie methods. The act method calls the transfer of the agent message with the effector commands. The think method between sense and act does the analysis of the perceptor data (e.g. a more elaborated world model) and the decision for plans and actions to be performed by the robot now and possibly in the future. The think method can of course be split into more dedicated deliberation methods which may be organized hierarchically if needed. All this can be worked out at the exercises during related courses. RoboNewbie provides just a simple example for illustration, the Agent_SimpleSoccer.

The Agent_SimpleSoccer is able to perform a very simple soccer play: As long as it is behind the ball and sees the opponent goal, it walks forward while pushing the ball with its feet. If the condition is not fulfilled, it turns around until it sees the ball, walks to the ball, turns around the ball until it sees the opponents goal, and then it starts walking towards the goal again. The decisions are made by a simple decision tree whenever the previous keyframe motion is completed (note that keyframe motions should not be interrupted).

Agent_SimpleSoccer can be improved in many ways. This is just what we want: Users can collect many ideas for improvements. They may concern better perception (e.g. by a ball model guiding the search), improved motions (like faster walk), new motions (like kick or dribble), better control (like path planning). It is possible to have more players on the soccer field such that players can cooperate (e.g. by positioning and passing). This gives room for simple contests during a course.

E. Logger

Runtime debugging of programs may be difficult because it affects synchronization with the server. Even simple debug messages printed on System.out may need too much time such that the agent cannot respond in time. It is possible to use the so-called sync mode which lets SimSpark RCSS wait until all agents have sent their messages (cf. the documentation). Alternatively, all debug messages can be collected by the program "Logger" of RoboNewbie. After the agent has finished, the collected messages are printed out. The usage is shown by the programs "Agent_TestPerceptorInput" and "Agent_TestLocalFieldView". Both programs provide also examples for the usage of the getter methods for perceptors.

VI. EXPERIENCES

The RoboNewbie framework was tested at different places. for introductory Robotics courses of about 30 hours during 5-8 days at Ohrid, Warsaw, Novi Sad, Rijeka, Sarajevo, and Plovdiv. 20 hours were planned for lectures, 10 hours for introduction and first usages of RoboNewbie. Additional 10-20 hours were used for further experiments by homework [9].

RoboNewbie served for illustrating experiments and for exercises in connection with the theoretical instructions. The participants of the courses learned to use RoboNewbie during short time and they programmed an improved soccer player at the end. The work with RoboNewbie was helpful to understand the theory. The final evaluation of the courses by the participants resulted in high marks. Especially the competitions with the improved soccer agent were motivating.

As RoboNewbie is intended for easy usage by beginners in Robotics, the requirements for the users are as minimal as possible, while the framework gives maximal support. For simplicity, approximations are used instead of complex calculations (e.g. simple offsets instead of linear algebra for determining the camera coordinates).

Since most of the courses had only a short duration, organisational issues were important for the success. We have asked the local organizers to prepare the technical resources accordingly. In the following, we describe some requirements in more detail.

A. Local Requirements for the Courses

Participants: Users are expected to have some programming skills in Java, such that they are able to understand and modify the agent programs. The programs are already prepared for usage under Netbeans, therefore the participants should be familiar with such tools. Users should be able to download and install programs from the web according to given instructions. Some physical and mathematical background is needed to understand the theoretical and practical issues of Robotics.

Participants should work in teams (as useful for programming exercises in general). Each team might consist of 3-5 participants, preferably mixed by different skills of its members. It helps for a smooth course if there are no big differences between the teams (e.g. each team should have at least one of the good programmers of the course, good mathematicians etc.).

Technical Resources: The participants should have their own computers where they can install and use the programs. Participants need access to the computers during the courses as well as for their homework. Hence, laptops are preferable. They are sufficient to run all the programs. Alternatively, participants may use computers in a lab (which have to be prepared accordingly if students are not allowed to install their own software).

The list of needed installations is given on the RoboNewbie webpage. Instructions for installation and functionality tests are

doi:10.15849/icit.2015.0012

found there, too. If possible, students should get information before starting the course. They should be asked to install the programs by themselves and test if the programs can be started. If students cannot be asked before, an on-site test by some responsible person should be performed. It helps to save time during the courses if on-site problems with hardware or software are solved before. Nevertheless, if computers are ready, installation of programs needs only short time and can be done at the beginning of a course.

Organisational Issues: A good schedule is necessary for smooth courses. This includes early information (as far as possible) of participants as described above. Then the lectures and exercises are mixed appropriately. After a short overview about Robotics, participants start their first exercises as given by the "Quick Start Tutorial" (also found on the RoboNewbie web page). Later, more explanations are given as far as the theoretical lectures proceed. Thus, theoretical introductions to sensors can be connected to explanations of perceptor usage in RoboNewbie, introductions to motions are connected to the development of keyframes etc.

B. Competition

Our courses end with a competition, which serves as a motivation for the participants. The successful participation at the competition can also be a substitute for an examination if students need some certificate.

The competition is announced at the beginning of the course, and it should be performed by the teams. This helps for the integration inside the teams from the very beginning. The number of competing teams should be not more than 10 in order to make the contest not too long. This is also an argument to form teams if the number of participants is larger. The level of teams should be comparable for fairness reasons.

Until 2014, the task for the competition was an improvement of the program Agent_SimpleSoccer to get a better performance. It was up to the teams, what they wanted to improve. Agent_SimpleSoccer performs very poorly as described before. It needs about 10 minutes to find the ball and to push it into the goal. It was designed this way just to motivate the participants for improvements.

To make the competition a success (and a fun), it must be organized by strict and transparent rules. It should have a tight schedule to emphasize the aspects of sports. Therefore, each team has only one trial of only 3 minutes. The ranking of teams is determined by fastest scoring times. For teams who did not score, the ranking is given by minimal distances to the goal after the 3 minutes have elapsed.

At the competition, each team gives a short description of its efforts and expected results. This is also a possibility to check the engagement of each team member. Moreover, each participant can be asked to provide a written report of his/her individual efforts.

At the course in Plovdiv 2014, a student group implemented a very powerful kick which allowed scoring immediately. Thus this kind of task is considered to be solved finally (in times of internet, copying this solution could make following competitions too simple). Thus, a new kind of competition with penalty kicks (attacker vs. goalkeeper) is tested in our recent courses.

C. Evaluation and Results

The participants of the courses were asked to give feedback on a prepared form at the end of the course. They could evaluate different aspects of the course and the framework.As the evaluation shows, the exercises with the simulated robots were motivating and helpful, the participants wanted to have more time for exercises and especially for own experiments.

As expected, the participants with less experience in Robotics gave higher marks related to motivation and help. The usage of the framework was intuitive. Interestingly, the participants with more experience in Java programming gave significantly higher rankings to the framework. The level of the exercises was considered as adequate, but for that the proportion of exercises was adapted by us accordingly.

As a unique observation, participants wanted to have more time for exercises than for lessons. This may have several reasons. The individual work load resulted in a bias for exercises: The participants had to fulfill given requirements, and many of them spent much time for preparing the final competition. Furthermore, the lectures tried to give a broad overview about the actual state of art in Robotics. There was not enough time to exercise on all these topics.

The "minimalistic approach" is useful especially for short courses and for introductions to longer courses. Later on, the disposability of non-minimalistic more sophisticated methods could be useful for higher level integrative tasks. It is impossible to let students implement all desirable algorithms in the limited time of a course. Joint activities of robots, for example, depend heavily on the available bodily skills and on the capabilities for interaction and coordination.

VII. CONCLUSION

The RoboNewbie framework can be used without special hardware. It simply needs a computer for simulation of the robot soccer scenario. The soccer scenario with humanoid robots is more complex than experiments by many hardware equipments. Nevertheless, RoboNewbie is easy to understand and to use after a short introduction. No special knowledge (except basic programming in Java) is required to start with own experiments, and while the users acquire more knowledge, they can work on more challenging tasks.

The practical evaluations have confirmed our expectations on the RoboNewbie project. Beginners in Robotics were able to use the framework after short introductions. They were able to program own methods in parallel to the theoretical concepts and methods provided by classes.

ACKNOWLEDGMENT

The first version of RoboNewbie was developed by Monika Domańska from the NaoTeam Humboldt [7]. We are thankful to the whole RoboCup community, especially to the developers

doi:10.15849/icit.2015.0012

of SimSpark RCSS, to the team magmaOffenburg and to our team NaoTeam Humboldt, and especially to Yuan Xu.

REFERENCES

[1] T. Padir, and S. Chernova (eds.), Special issue on robotics education. IEEE Transactions on Education, vol. 56, issue 1, 2013. http://ieeexplore.ieee.org/xpl/tocresult.jsp?isnumber=6423944. Visited at 19.1.2015.

[2] Yuan Xu**,** From Simulation to Reality: Migration of Humanoid Robot Control. Dissertation Humboldt University Berlin, 2013.

[3] RoboCup Web page. http://www.robocup.org. Visited at 19.1.2015.

[4] Aldebaran Web presence http://www.aldebaran-robotics.com/en/. Visited at 19.1.2015.

[5] SimSpark RCSS Wiki (Documentation of the Simulator). http://simspark.sourceforge.net/wiki. Visited at 19.1.2015.

[6] R. Smith, Open Dynamic Engine User Guide, 2006. http://www.ode.org. Visited at 19.1.2015.

[7] RoboNewbie. http://www.naoteamhumboldt.de/projects/robonewbie/. Visited at 19.1.2015.

[8] Homepage Team magmaOffenburg. http://robocup.hs-offenburg.de/. Visited at 19.1.2015.

[9] M. Domańska, H.D. Burkhard, RoboNewbie: A Framework for Experiments with Simulated Humanoid Robots. In M. Ivanović, L.C. Jain (eds.), E-Learning Paradigms and Applications, Agent-based Approach. Springer Series: Studies in Computational Intelligence, vol. 528, 2014, pp. 1-38.