Noxious-Kouretes 2011 SPL Team Description Paper*

E. Chatzilaris[3], I. Kyranou[3], J. Ma[2], E. Orfanoudakis[3], A. Panakos[1], A. Paraschos[1], G. Pierris[1], N. Spanoudakis[3], J. Threlfall[1], A. Topalidou[3], D. Tzanetatou[3], E. Vazaios[3], S. Cameron[2], T. Dahl[1], M. G. Lagoudakis[3]

[1] Cognitive Robotics Research Center (CRRC), University of Wales, Newport, UK
crrc.newport.ac.uk

[2] Computing Laboratory (ComLab), Oxford University, Oxford, UK
www.comlab.ox.ac.uk

[3] Intelligent Systems Laboratory, Technical University of Crete, Chania, Greece
www.kouretes.gr

1 Team Information

Team Noxious-Kouretes is a joint team formed by three universities (Newport, Oxford, TUC) in two countries (UK, Greece). This collaboration emerged naturally following a recent RoboCup event (RoboCup Exhibition and Engagement Event, Eisteddfod of Wales 2010) where all three groups met and a recent move of personnel between groups. The minimal overlap over the research areas of the three groups further motivated the joining of forces towards a strong and competitive joint team, building on each group's history of RoboCup participation.

Team Noxious is a collaboration between the Cognitive Robotics Research Center (CRRC), University of Wales, Newport and the Computing Laboratory (ComLab), University of Oxford. The team was formed only in May 2010, but soon after, in August 2010, the team organized and hosted UK's first official RoboCup event RC4EW, featuring Standard Platform and 3D Simulation leagues. Two members of Noxious team are part of the newly created NAOShare community which consists of 30 European professors and students sharing academic resources engaging the Nao robot in educational programs. The team has received several distinctions, mostly in the 2D/3D Sim League under the name OxBlue: **runner-up** in UK FIRA competition 2007; **8th** place in 2D Sim, **12th** place in 3D Sim at RoboCup 2008; **2nd** place in 2D Sim at RoboCup 2009; **3rd** place in 3D Sim at RC4EW 2010; and **4th** in SPL at Iran Open 2011 SPL.

Team Kouretes was founded in 2006 and participates in the main RoboCup competition ever since in various leagues (Four-Legged, Standard Platform, MSRS, Webots), as well as in various local RoboCup events (German Open, Mediterranean Open, RC4EW, RomeCup) and RoboCup exhibitions (Athens Digital Week, Micropolis, Schoolfest). In May 2010, the team hosted the 1st official SPL tournament in Greece (with three invited teams) within the Hellenic Conference on Artificial Intelligence (SETN). The team has been developing its own (publicly-available) software for the Nao robots since 2008. Distinctions of the team include: **2nd** place in MSRS at RoboCup 2007; **3rd** place in SPL-Nao,

* Team Kouretes has been supported by the Technical University of Crete, the European Grant MCIRG-CT-2006-044980, and Chipita S.A.–Molto (exclusive sponsor).

1st place in SPL-MSRS, among the **top 8** teams in SPL-Webots at RoboCup 2008; **1st** place in RomeCup 2009; **6th** place in SPL-Webots at RoboCup 2009; and **2nd** place in SPL at RC4EW 2010.

Team Noxious-Kouretes is jointly led by Stephen Cameron (reader), Torbjørn S. Dahl (reader), and Michail G. Lagoudakis (assistant professor). The 12 members of the team for 2011 (in alphabetical order) and their research areas are:

Eleftherios Chatzilaris	grad student	(TUC)	[Localization, Learning]
Iris Kyranou	ugrad student	(TUC)	[Obstacle Avoidance]
Jie Ma	postdoc	(Oxford)	[Walk Learning]
Emmanuel Orfanoudakis	ugrad student	(TUC)	[Visual Object Recognition]
Andreas Panakos	grad student	(Newport)	[Color Recognition]
Alexandros Paraschos	lab staff	(Newport)	[Software Architecture]
Georgios Pierris	grad student	(Newport)	[Motion Skill Learning]
Nikolaos Spanoudakis	lab staff	(TUC)	[ASEME Methodology]
John Threlfall	ugrad student	(Newport)	[Object Recognition]
Aggeliki Topalidou	ugrad student	(TUC)	[Behavior, Coordination]
Dimitra Tzanetatou	ugrad student	(TUC)	[Motion Skill Management]
Evangelos Vazaios	grad student	(TUC)	[Robot Communication]

2 Team Research

2.1 Software Architecture and Communication (TUC/Newport)

The team's code is based on MONAS [4], a software architecture developed in-house to address the needs of the team. MONAS provides an abstraction layer from the Nao robot and allows the synthesis of complex robotic teams in various ways: (a) NaoQi modules, (b) XML-specified MONAS agents, and (c) model-based (ASEME) statechart agents. Apart from the trivial method (a), method (b) allows the coding of different agents (e.g. perception, motion, behavior, etc.) each executed independently at any desired frequency completing a series of activities at each execution. Method (c) is based on the Agent Systems Engineering Methodology (ASEME), whereby the target team behavior is specified through a series of model-based transformations as a statechart, which is executed using a generic multi-threaded statechart engine. MONAS is shown in Figure 1 (left).

To realize a fully-functional robot team, MONAS relies on our intra-robot and inter-robot messaging system, called NARUKOM [9]. NARUKOM is based on the publish/subscribe paradigm and provides maximal decoupling not only between nodes, but also between threads on the same node. The data shared between nodes/threads are stored on local blackboards which are transparently accessible from all nodes/threads. Communication is achieved through messages tagged with appropriate topics and relayed through a message queue which is implemented using Google protocol buffers. Three types of messages are supported: (i) *state messages*, which remain in the blackboard until they are replaced by a newer message of the same type, (ii) *signal messages*, which are consumed at the first read, and (iii) *data messages*, which are time-stamped to indicate the precise time the values they carry were acquired. Data messages have expiration

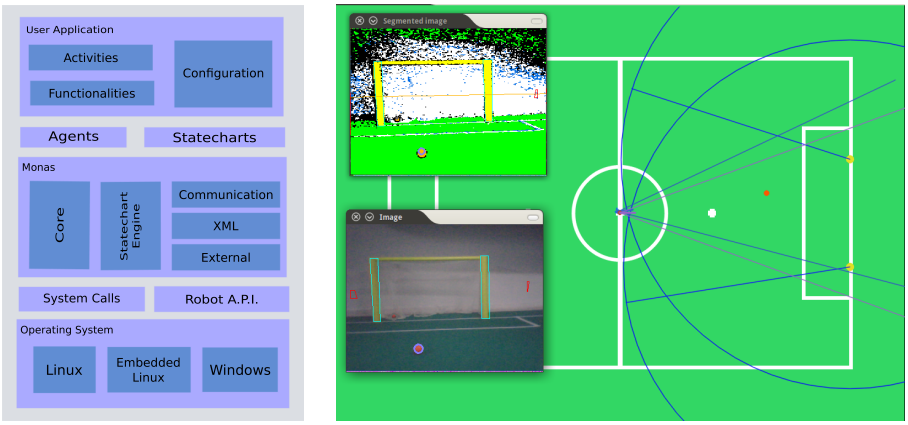

Fig. 1. Monas Architecture (left) and Perception: raw image, color-segmented image, recognized objects, localization GUI (right).

times, so that they are automatically removed when they are no longer needed. NARUKOM's distributed nature and platform independence make it an ideal base for coordinating asynchronous modules, for developing complex team strategies, and for relaying debugging information to external computers.

2.2 Vision and Localization (TUC/Newport)

The objects of interest in the SPL are characterized by unique colors. Our approach to color recognition is based on labeling by hand a representative set of images from the robot camera, training a classifier which generalizes over the entire color space, and generating a static color map for constant-time recognition during deployment. This process is supported by our KC^2 graphical tool [3]. Since light sources affect significantly the correct recognition of colors, the white and gray surfaces of the Nao body are utilized for providing a reference point for the illumination of the environment and for calibrating the camera's white balance dynamically. This dynamic camera calibration enables the use a single color map (trained at RoboCup 2010) in a wide range of lighting environments.

Object recognition is accomplished by KVISION, a light-weight image processing scheme which applies color recognition to a selected set of pixels directly on the YUV images provided by the hardware to locate areas of interesting colors. These areas are then processed locally and examined further for validity as field objects. The exact camera position in the 3D space (extracted from the robot's kinematic chain) is utilized to determine the sampling grid, so that scanning is uniformly projected over the field, not over the image matrix. Special attention is paid in synchronizing precisely images with robot joint values using time-stamped data messages. Kinematics and camera position are also used to determine the view horizon, the expected projected size of the known-size SPL objects on the image, and the head joint values for fixating on objects of interest. Multiple matches for various objects (e.g. multiple balls) are evaluated and filtered, so that the best match (if any) in terms of color, shape, and size for

each unique object is finally extracted and returned as perceived, along with an estimate of its distance and bearing as well as head joint information for fixation and tracking. Figure 1 (right) shows an example of visual object recognition.

Self-localization of the robots in the field is accomplished using KLoc [1] which realizes Monte Carlo localization with particle filters. The belief of the robot is a probability distribution over the 3D space of (x, y, θ), where x, y are the robot coordinates on the field from a bird's eye view and θ is the orientation of the robot with respect to the line that crosses the center of the goals. The belief is represented approximately using a population of particles. In order to perform belief update, we use an odometry motion model that covers all possible omni-directional locomotion choices and a landmark sensor model for the yellow/blue goalposts over the (x, y, θ) space. The parameters of both models were estimated through extensive experimentation. Belief update is performed using an auxiliary (AUX) particle filter. For the resampling process, selection with replacement and linear-time resampling have been implemented. Given any population of particles, the robot's pose is estimated as the pose of the particle with the highest weight. Figure 1 (right) shows our localization GUI which displays the perceived landmarks (two yellow goalposts) and the estimated robot position.

2.3 Walk Learning (Oxford)

Our QWalking method [2] is a careful application of the popular Q-learning algorithm to the problem of acquiring robot walking patterns. The advantage of using a learning technique over analytical methods is twofold: we can deal with imprecise models of the robot's dynamics or kinematics; and we can adjust the learned actions as the robot joints wear out. The major difficulty in applying Q-learning to this task is the large size of the state space and thus the slow speed of learning convergence. We deal with this problem by using a simplified gait model with a small number of parameters and by first applying learning within a physically accurate simulation of the robot (the SPARK simulator). The former yields a state space of manageable size (a few thousand states), while the latter greatly decreases the overall learning time, not least because the robot is usually dynamically stable when learning switches to the real robot. Finally, through incremental Q-learning, we can refine the walking patterns as the robot ages. QWalking has already been used for producing walking patterns in the 3D Sim league. Figure 2 shows a description of the leg swing phase (left) and an example of a continuous gait pattern generated by QWalking (right). For SPL, we transferred the refined fast walking patterns from the 3D Sim directly onto the real Nao robots. The speed of QWalking is approximately 22 cm/sec, which roughly doubles the original Aldebaran walking speed (11.3 cm/sec). Our plans include omnidirectional walking, dynamic balancing, and energy-efficient gaits.

2.4 Motion Skill Learning (Newport/TUC)

Motion skills, such as player kicking or goalkeeper diving, are crucial for a RoboCup team. The majority of teams use manually-designed, pre-recorded motions, which are fine-tuned on spot for optimized performance. Despite the

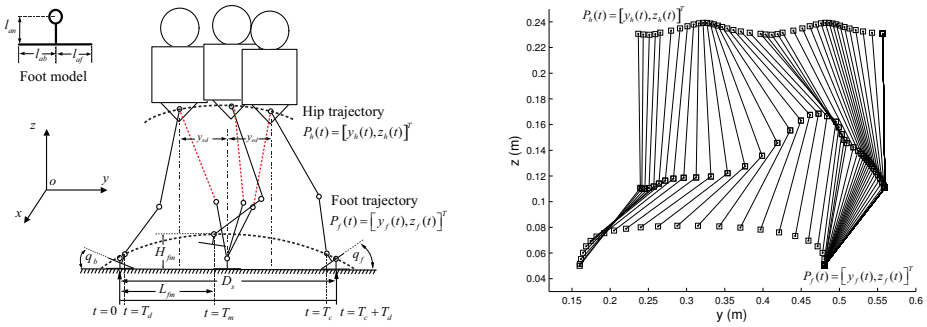

Fig. 2. Description of leg swing phase during walk (left) and a learned continuous gait pattern generated by QWALKING (right).

drawback of these approaches, automatic skill acquisition/learning is not widely used, due to long learning periods and robot durability/safety issues. Our team currently makes a transition from manual motion design (e.g. our past work on the KME tool [6]) to learned motions focusing on biologically-inspired approaches to learning gestures and motion skills on humanoid robots. Our algorithms are based on hierarchical temporal structuring of states and actions using fixed-length data sequences on each level of the hierarchy. Hierarchy is achieved using hierarchical self-organizing maps (SOMs), and temporally is achieved using decaying activity levels for winning SOM nodes. This structure massively compresses temporal data through the reuse of common sub-sequences in the hierarchy and through a principled component-based representation of each fixed sequence. We have successfully used this structure to learn and reproduce humanoid motions in our CoSCo-HSOM algorithm [5]. We have also made progress towards applying reinforcement learning in such structures by successfully demonstrating mechanisms for future reward prediction and hidden state identification in hierarchies of fixed size temporal sequences. The current version of CoSCo-HSOM does not use any feedback from the environment, limiting its capabilities to only creating open loop motions, which may not fit all SPL needs.

We are also actively working on fine-tuning closed-loop motion controllers. Our robots need to maintain their balance while performing some complex action, such as a kick, which requires balancing on one leg. We consider the problem of learning the parameters of a closed-loop controller that controls only the ankle joints of the leg (roll and pitch) based on dynamic (real-time) information from the FSRs and the inertial unit with the goal of keeping the robot balanced. Such a skill eliminates the need for costly kinematic computations and allows all other joints to perform arbitrary moves without worrying about balancing. So far, we have succeeded in maintaining balance on a single leg when the other joint chains of the robot move to any arbitrary angle, however at a slow speed. Our current efforts focus on Monte-Carlo EM-type reinforcement learning methods [10] for optimizing our current PD controller, aiming eventually at real-time operation at any motion speed and robustness over the entire range of balancing cases.

2.5 Motion Skill Management (TUC)

Walking speed and special action completion times are crucial in SPL games. We have observed that significant time is lost in gradually approaching the ball, stopping to take the right pose, and then executing an appropriate kick action. In order to decrease the total response time, the interleaving of walking and special actions, specifically kick actions, was studied. The idea is to kick the ball directly, without slowing down while approaching the ball. The `kick-forward` special action was analyzed and some of the leg poses that comprise the kick action were bookmarked as potential entry points into the kick sequence, along with the intended direction of change (given by the next pose in the action sequence). As the robot walks, the 6 most significant out of its 11 leg joints, in particular Hip Roll, Hip Pitch, and Knee Pitch of each leg, are constantly monitored in terms of their current joint values and their current gradients (directions of change). If the current leg joint pose (only the selected 6 poses) comes close to one of the bookmarked poses in terms of both joint values and directions of change, then the robot switches immediately from walking to the corresponding entry point of the kick special action and executes the kick. This way, the kick is executed before the robot stops walking. Obviously, this transition is enabled only when the robot is close enough to the ball. This simple technique has significantly reduced the total time of approaching and kicking. However, the entry poses must be chosen carefully to ensure the stability of the robot during the sudden transition from walking to kicking and the kick precision is sometimes compromised.

2.6 Obstacle Avoidance (TUC)

The SPL rules postulate severe penalties for player pushing, therefore we have dedicated considerable effort in avoiding collisions. Because of the limited coverage and the noisy readings of the Nao ultrasonic sensors, each robot builds and maintains a local obstacle map of its surrounding. This map is stored in a $n \times k$ polar grid (currently, 7×36, giving 10cm and 10° resolution respectively) covering the 360° area of radius 70cm around the robot. The robot is always located at the center of the grid and the grid moves with the robot. This polar map is updated using distance readings from the ultrasonic sensors. The polar topology facilitates the mapping of the sensor data and also captures the resolution of the sensors which is dense close to the robot and sparse away from the robot due to the conical shape of the beams. Visually perceived obstacles can also be mapped on the same grid. Locomotion of the robot implies appropriate transformations on the polar grid. Rotational and translational motions, as captured by the locally-accurate odometry, are reflected on the map in constant time through precomputed and stored transformations, albeit with some loss of accuracy due to discretization. Each cell in the map stores the probability that there is an obstacle at the corresponding position in the field. These values are updated according to a simple sensor model (for cells within the sensor cone) or according to an aging model (for cells outside the sensor cone). The polar obstacle map is used to plan obstacle-free paths in any desired direction of motion or to any desired destination. A simple example is demonstrated in Figure 3.

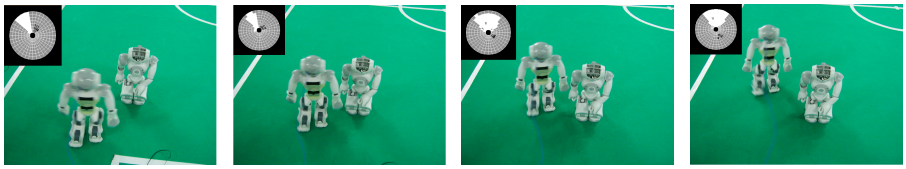

Fig. 3. Obstacle avoidance with the polar obstacle map (robot front is towards the top of the map and gray levels indicate the occupancy probability, low-white, high-black).

Role: player
Protocols: Attack: center, Attack: center_for
Liveness:
player = Sensors$^{\omega}$ || RobotController$^{\omega}$ || LedHandler$^{\omega}$ || MotionController$^{\omega}$ || (initialize . activate)
initialize = Stand . CalibrateCamera
activate = Vision$^{\omega}$ || Localization$^{\omega}$ || ObstacleAvoidance$^{\omega}$ || HeadHandler$^{\omega}$ || decision$^{\omega}$
decision = CheckForBallObservation . (ScanForBall | action)
action = TrackBall || (WalkTowardsBall | KickBall | center | center_for)
center = WalkTowardsBall . [PassBall]
center_for = WalkTowardsGoal . [WalkTowardsBall . [KickBall]]

Fig. 4. The SRM model for the `player` (roles `center` and `center_for`).

2.7 Behavior (TUC)

The highest level of decision making in our team is modeled as a statechart using a model-based process for agent engineering (ASEME) [8]. ASEME enables the designer to specify complex behaviors through a series of semi-automated design phases that produce the final statechart model which is then transformed automatically to source code, compiled, and executed using our multi-threaded statechart engine. The final statechart model specifies the intra- and the inter-agent control for achieving the desired behavior by accessing directly the functionality provided by our base activities: RobotController, Vision, Localization, ObstacleAvoidance, MotionController, HeadHandler, Sensors, LedHandler. Using this principled methodology, we redeveloped our simple Goalie and Player behaviors from RoboCup 2010, previously implemented as MONAS agents (conventional procedure code), in dramatically less time without debugging headaches. The added value of ASEME was revealed in the development of a team `Attack` protocol involving two players (`center` and `center_for` interchangeable roles), which was supported by the ability of ASEME to specify inter-agent control and the ability of MONAS/NARUKOM to realize it on the robots.

To briefly demonstrate our approach, consider an `Attack` protocol with two players assuming the roles of `center` and `center_for` (one role each). The player closer to the ball becomes the `center` and the other becomes the `center_for`. The `center` is expected to approach the ball and pass it to the `center_for` who, in turn, is expected to be near the opponent's goal to receive the pass and shoot to the goal. The protocol may terminate early, if an opponent takes control of the ball or the ball goes out of bounds. We use the Gaia operators for writing a System Roles Model (SRM) liveness formula that defines these two

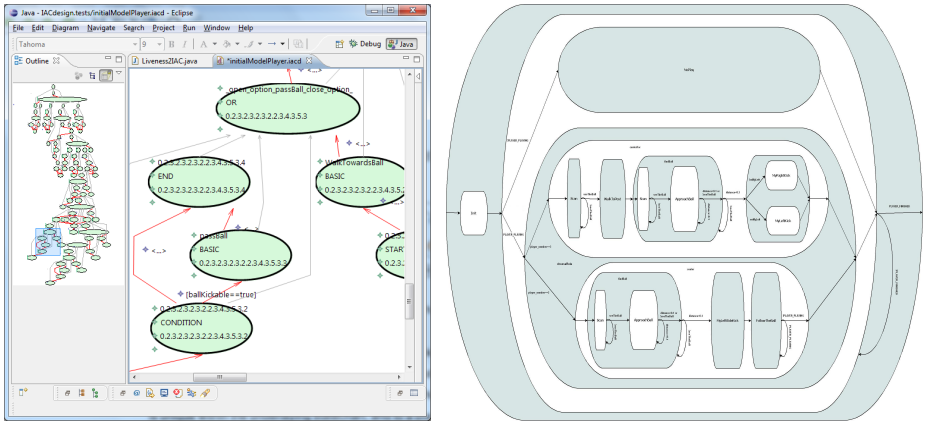

Fig. 5. The graphical editor (left) and the statechart (right) for the `player`.

roles, as shown in Figure 4. Briefly, $A.B$ means that activity B is executed after activity A, A^{ω} means that A is executed continuously, $A|B$ means that either A or B is executed, $A||B$ means that A and B are executed in parallel, and $[A]$ means that A is optional. Using the `SRM2IAC` tool [7] the SRM liveness formula (given in plain text) is automatically transformed to a statechart which can be edited using our graphical tool (Figure 5). Using the tool, we add the transition expressions and the variables related to each transition and node respectively. Finally, we use the `IAC2Monas` tool [4] for auto-generating the `C++` code.

References

1. Chatzilaris, E.: Visual-Feature-based Self-Localization for Robotic Soccer. Diploma thesis, Technical University of Crete, Greece (2009)
2. Ma, J., Cameron, S.: Learning robust and energy-efficient biped walking patterns using QWalking. In: Intl Conf on Climbing and Walking Robots (CLAWAR) (2009)
3. Panakos, A.: Efficient Color Recognition Under Varying Illumination Conditions for Robotic Soccer. Diploma thesis, Technical University of Crete, Greece (2009)
4. Paraschos, A.: Monas: A Flexible Software Architecture for Robotic Agents. Diploma thesis, Technical University of Crete, Greece (2010)
5. Pierris, G., Dahl, T.: Compressed sparse code hierarchical SOM on learning and reproducing gestures in humanoid robots. In: IEEE Intl Symposium on Robot and Human Interactive Communication (RO-MAN) (2010)
6. Pierris, G.F., Lagoudakis, M.G.: An interactive tool for designing complex robot motion patterns. In: IEEE Intl Conf on Robotics and Automation (ICRA) (2009)
7. Spanoudakis, N., Moraitis, P.: Gaia agents implementation through models transformation. In: Intl Conf on Principles of Practice in Multi-Agent Systems (2009)
8. Spanoudakis, N., Moraitis, P.: Using ASEME methodology for model-driven agent systems development. In: Intl Workshop on Agent-Oriented Software Engineering (AOSE) (2010), www.amcl.tuc.gr/aseme
9. Vazaios, E.: Narukom: A Distributed, Cross-Platform, Transparent Communication Framework. Diploma thesis, Technical University of Crete, Greece (2010)
10. Vlassis, N., Toussaint, M., Kontes, G., Piperidis, S.: Learning model-free robot control by a Monte Carlo EM algorithm. Autonomous Robots 27(2), 123–130 (2009)