Kouretes 2012 SPL Team Description Paper*

Maria Karamitrou, Nikolaos Kofinas, Nikolaos Pavlakis,
Angeliki Topalidou-Kyniazopoulou, Astero-Dimitra Tzanetatou,
Nikolaos I. Spanoudakis, Michail G. Lagoudakis

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

1 Team Information

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, Iran 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, **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; **2nd** place in SPL at RC4EW 2010; and **2nd** place in SPL Open Challenge Competition at RoboCup 2011 (joint team Noxious-Kouretes).

Team Kouretes is led by Michail G. Lagoudakis (assistant professor) and Nikolaos I. Spanoudakis (laboratory staff). The members of the team are senior undergraduate ECE students; their names and research areas are listed below:

Angeliki Topalidou-Kyniazopoulou	[CASE Tool for Statechart Design]
Dimitra-Astero Tzanetatou	[Motion Skill Management]
Maria Karamitrou	[Monitoring and Debugging Tools]
Nikolaos Kofinas	[Forward and Inverse Kinematics]
Nikolaos Pavlakis	[Formations and Team Strategy]

2 Team Research

2.1 Software Architecture and Communication

The team's code is based on MONAS [5], 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 as XML-specified Monas modules and/or statechart modules. Monas modules focus on specific functionalities (vision, motion, etc.); they are executed concurrently, each one of them at any desired frequency completing a series of activities at

* 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).

each execution. Statechart modules are based on the Agent Systems Engineering Methodology (ASEME) [8], whereby the target team behavior is specified through a series of model-based transformations as a statechart. Statecharts are executed using a generic multi-threaded statechart engine that provides the required concurrency and meets the real-time requirements of the activities on each robot. A high-level view of MONAS is shown in Figure 1 (left).

MONAS relies on our intra-robot and inter-robot messaging system, called NARUKOM [12]. NARUKOM is based on the publish/subscribe paradigm and supports multiple ways of communication, including point-to-point and multicast connections. 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 implemented using Google protocol buffers to facilitate the serialization of data and the structural definition of the messages. Three types of messages are supported: (i) *state*, which remain in the blackboard until they are replaced by a newer message of the same type, (ii) *signal*, which are consumed at the first read, and (iii) *data*, which are time-stamped to indicate the precise time the values they carry were acquired. Data messages have expiration times, so that they are automatically removed when they are no longer needed.

2.2 Vision and Localization

The objects of interest in the SPL are characterized by unique colors. Our method for 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 [4]. 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 of a single color map in a wide range of lighting environments.

Object recognition is accomplished by KVISION [3], a light-weight image processing method for visual object recognition on articulated mobile robots with a head-mounted camera, such as humanoid robots, focusing on reliability and efficiency. The vision pipeline uses real-time sensory information from the joints along with the robot's kinematic chain to determine the exact camera position in the 3-dimensional space and subsequently the view horizon and the sampling grid, so that scanning is approximately uniformly projected over the ground (field), not over the image matrix. Special attention is paid on the precise synchronization of images with robot joint values using time-stamped data messages. The next phase employs our auto-calibrated color recognition scheme on the pixels of the sampling grid to identify regions of interest. In the last phase, detailed analysis of the identified regions of interests seeks potential matches for the corresponding target object. These matches are evaluated and filtered by several heuristics, so that the best match (if any) in terms of color, shape, and

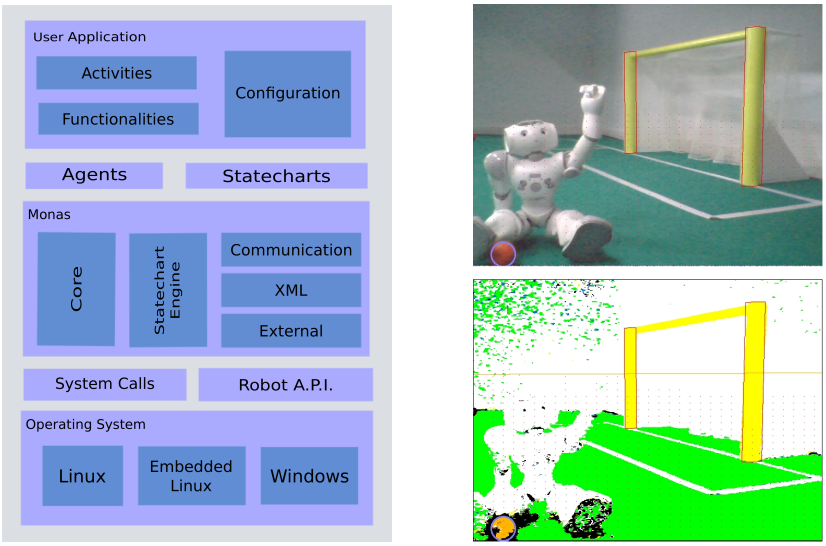

Fig. 1. MONAS software architecture (left), KVISION object recognition (right).

size for a target object is finally extracted. Then, the corresponding object is returned as perceived, along with an estimate of its current distance and bearing as well as appropriate head joint angles for fixating and tracking it. Figure 1 (right) shows an example of ball and goal post recognition.

Self-localization is accomplished using KLOC [1] which realizes Monte Carlo localization. The belief of the robot is a probability distribution over the 3-dimensional space of (x, y, θ) (coordinates and orientation) represented approximately using a population of particles. Belief update is performed using an auxiliary particle filter with an odometry motion model for omnidirectional locomotion and a landmark sensor model for the goalposts (landmarks). The robot's pose is estimated as the pose of the particle with the highest weight.

2.3 Monitoring and Debugging

To facilitate software development, we have developed a monitoring and debugging tool, call KMONITOR, which monitors messages sent by the robots over the network using the UDP packet multicasting protocol and displays the transmitted information in a user-friendly way. The user is able to visualize the world state of each robot connected to the network, namely its estimated position in the field, the estimated location of the ball with respect to itself, the current ball and goal observations, the particles forming its current belief, a trace of the most recent positions, the current field of view, etc. Figure 1 shows a simple example, where KMONITOR displays the position and ball estimates of two robots. Note that perception and localization errors lead in perceiving the same ball as being at slightly different locations on the field. We are currently working on a shared world model, whereby the robots will be able to share and fuse world information. KMONITOR has been implemented using Qt and is parameterized through

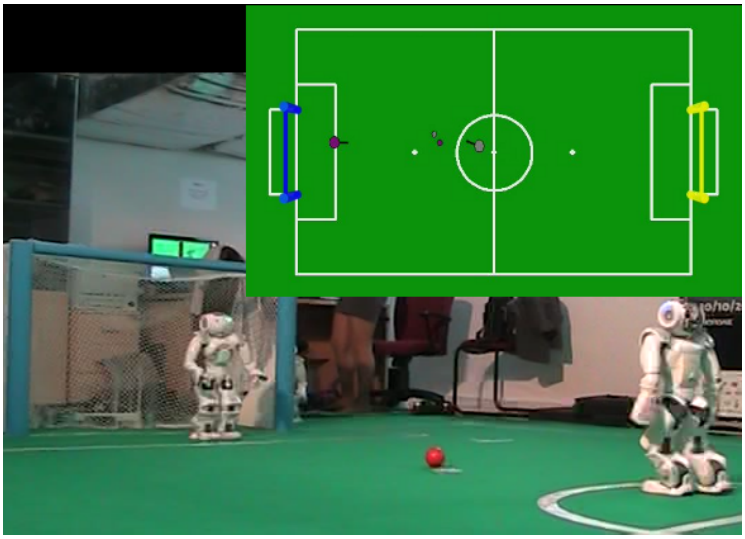

Fig. 2. KMonitor graphical user interface for monitoring and debugging.

an XML file for different field configurations (field size, location and size of goals, location of lines, etc.). Our current work focuses on extending KMonitor to a fully integrated tool, which allows the developer to monitor all the important information of each robot, including camera images, obstacle map, player state, and behavior decisions visually in real-time.

2.4 Kinematics

We have completed our own analytical derivation and implementation of the forward and inverse kinematics for the whole body of the Nao robot (head, arms, legs). As an example, we list the equations we have derived for the inverse kinematics of the left arm. Let (x, y, z) be the coordinates of the left arm base point with respect to the point of reference (the center of the robot torso), (p_x, p_y, p_z) be the desired position (Cartesian coordinates) of the left hand, and (a_x, a_y, a_z) the desired orientation (Eulerian angles) of the left hand. If l_1 is the length of the upper arm and l_2 is the total length of the lower arm plus the offset of the (passive) hand, then the target arm configuration $(\theta_1, \theta_2, \theta_3, \theta_4)$ giving the angles of the LShoulderPitch, LShoulderRoll, LElbowYaw, and LElbowRoll joints can be computed as follows:

$$\theta_4 = \pi - \arccos\left(\frac{l_1^2 + l_2^2 - \Big((x-p_x)^2 + (y-p_y)^2 + (z-p_z)^2\Big)}{2l_1l_2}\right)$$

$$\theta_2 = \pm\arccos\left(\frac{p_y - l_3 - l_1\Big(\sin a_x \sin a_y \sin a_z + \cos a_x \cos a_y\Big)\dfrac{\sin\theta_4}{\cos\theta_4}}{l_2 + l_1\cos\theta_4 + l_1\dfrac{\sin^2\theta_4}{\cos\theta_4}}\right) + \frac{\pi}{2}$$

$$\theta_3 = \arcsin\left(\frac{\sin a_z \sin a_y \cos a_x - \cos a_z \sin a_x}{\sin\left(\theta_2 - \frac{\pi}{2}\right)}\right)$$

$$\theta_3 = \pi - \arcsin\left(\frac{\sin a_z \sin a_y \cos a_x - \cos a_z \sin a_x}{\sin\left(\theta_2 - \frac{\pi}{2}\right)}\right)$$

$$\theta_1 = \pm \arccos\left(\frac{\cos a_y \cos a_x - \left(\cos a_z \sin a_y \cos a_x + \sin a_z \sin a_x\right)\frac{\sin\theta_3 \cos\theta_2}{\cos\theta_3}}{\cos\theta_3 + \frac{\cos^2\theta_2 \sin^2\theta_3}{\cos\theta_3}}\right) \text{ if } \theta_3 \neq \frac{\pi}{2}$$

$$\theta_1 = \pm \arccos\left(\frac{\cos a_z \sin a_y \cos a_x + \sin a_z \sin a_x}{\cos\theta_2 \sin\theta_3}\right) \qquad \text{if } \theta_3 = \frac{\pi}{2}$$

Due to symmetries, these inverse kinematics equations lead to a small number of distinct solutions, some of which may be infeasible or invalid because of the constrained range of each joint. To determine a valid and correct solution, we simply run each one of these solutions through the forward kinematics to verify that indeed the end point of the hand reaches the desired position and orientation. The equations for the right arm are similar.

Kinematics are currently employed to enhance the ability of vision. In particular, forward kinematics are used to determine the exact camera position in the physical space given the readings from the joint encoders. This allows the precise synchronization of a camera image with the exact camera position given the joint angles at the very moment the image was obtained; this functionality is not available through the ALMotion module of NaoQi, which provides forward kinematics only for the current joint configuration. The synchronization leads to significant improvements in estimating the distance and bearing of recognized objects. In addition, inverse kinematics of the head are used to point the camera to any desired field position; this feature is particularly useful in turning the head towards an estimated ball position after a wide scan for landmarks, while walking towards the ball.

2.5 Motion Skill Management

Walking speed and special action completion times are critical factors in SPL games. It has been observed that significant time is lost in approaching the ball, stopping to assume the right pose, and then executing an appropriate kick action. In order to decrease the total response time, we have developed a method for interleaving walk and kicks [11], aiming at kicking the ball directly without slowing down while approaching it. Our (forward, side, and back) kick actions are designed using Kouretes Motion Editor (KME) [7] and consist of timed sequences of poses executed at the DCM level. Therefore, the leg poses of our

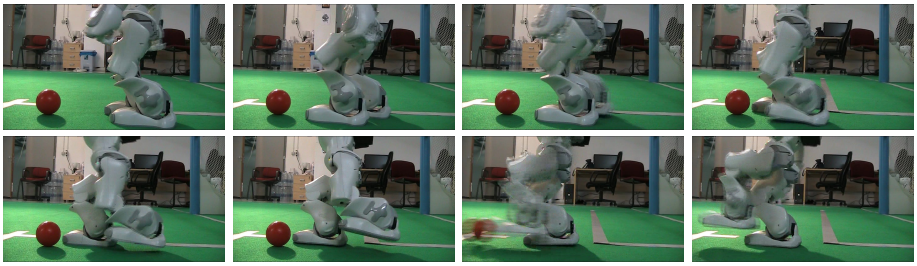

Fig. 3. Interleaving of walk and forward kick (order: left to right, top to bottom).

kick actions were analyzed against the recorded walk leg poses (captured dynamically from a walking robot) to identify close matchings and some of them were bookmarked as potential entry points into the kick sequences. This analysis involved the 8 most significant out of the 11 leg joints, namely Hip Roll, Hip Pitch, Knee Pitch of each leg, and Ankle Pitch and Ankle Roll for the balancing leg, as well as their gradients (rate of change), and the Euclidean (L_2) norm over these 16-dimensional vectors as a similarity metric. As the robot walks up to the ball and gets to kicking distance, these features are monitored in real-time. As soon as the current walk pose comes close to one of the bookmarked poses of the desired kick (norm below a threshold), the robot switches immediately from walking to the corresponding entry point and executes the kick sequence from that point. 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. Our implementation is fully parameterizable through XML files to accommodate any special actions, walk gaits, and similarity metrics. Our motion interleaving technique has significantly reduced the total time of approaching and kicking the ball, yielding significant advantage to our robots. However, the entry poses must be chosen carefully to ensure the stability of the robot during the sudden transition and the precision of the kick. Figure 3 shows an example of performing a forward kick interleaved with walk.

2.6 Obstacle Avoidance

Path planning decisions of a robot are important in avoiding collisions in the dynamic environment of a populated SPL field, since such collisions cause hardware damages, loss of valuable time, and penalties according to the league rules. We have developed a path planning method for autonomous robots in dynamic environments [2], which relies on the construction of a local egocentric occupancy polar map for representing obstacles and the A* heuristic search algorithm for deriving collision-free paths to desired goals. Our approach creates and maintains a probabilistic occupancy grid map which discretizes the physical area close and around the robot using variable resolution and holds the robot's belief about the existence or not of obstacles. This map is stored in a $n \times k$ polar grid (currently, 10×18, giving 10cm and 20° resolution respectively) covering the 360° area of radius 100cm around the robot. 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. Each cell in the map stores the probability that there is an obstacle at the corresponding position in the field. The occupancy values in the map are updated according to a simple sensor model (for cells within the sensor cone) using real-time sensory information from ultrasonic or any other range sensors or according to an aging model (for cells outside the sensor cone). The robot is always located at the center of the grid and the grid moves with the robot. Translation and rotation transformations of the map are implemented, in order to maintain the correct placement of obstacles relative to the robot consistently with its movements. Given a desired target position and orientation, an A* heuristic search algorithm in the three-dimensional space (coordinates on the field and orientation) derives an optimal (with respect to the grid) path taking into account the omni-directional locomotion abilities of the robot (forward, side, rotating steps and combinations). The derived path provides the motion controller with the waypoints needed to set appropriately the velocities of the robot to move along the path. This method enables our robots to move safely around the field without collisions towards any desired position and orientation.

2.7 Behavior Specification

The behavior of our team is specified using the Agent Systems Engineering Methodology (ASEME) [8] as it was recently adapted to address the challenges of robotic behavior specification [6]. ASEME suggests a strict hierarchical decomposition of the desired robot behavior into smaller activities until a level of provided base activities is met. Each design step is supported by a formal model. Following a model-driven engineering approach, the transition from one design phase to another is assisted by automatic model transformations leading from requirements to computer programs. Briefly, the process begins with the specification of a set of liveness formulas (analysis phase) which are converted to an initial statechart model; the statechart is subsequently enriched (design phase) and is converted to source code.

Our own Computer-Aided System Engineering (CASE) tool, Kouretes Statechart Editor (KSE) [9,10], enables the developer to design the statechart model either graphically, from scratch, or by first writing the liveness formulas and transforming them automatically to an abstract statechart. KSE also supports the graphical editing of the statechart model, the addition of transition expressions, the validation of the final statechart model indicating which elements are incorrect (if any), and the automatic source code generation for compilation and execution on the robot. The final statechart model specifies the intra- and the inter-agent control for achieving the desired team behavior by accessing directly the functionality provided by our base activities: RobotController, Vision, Localization, ObstacleAvoidance, MotionController, HeadHandler, Sensors, LedHandler, Communication. Figure 4 shows a screenshot of KSE with the complete statechart for a simple single-robot behavior: *sit down when you see the ball and track it, stand up and scan for the ball when you lose it.*

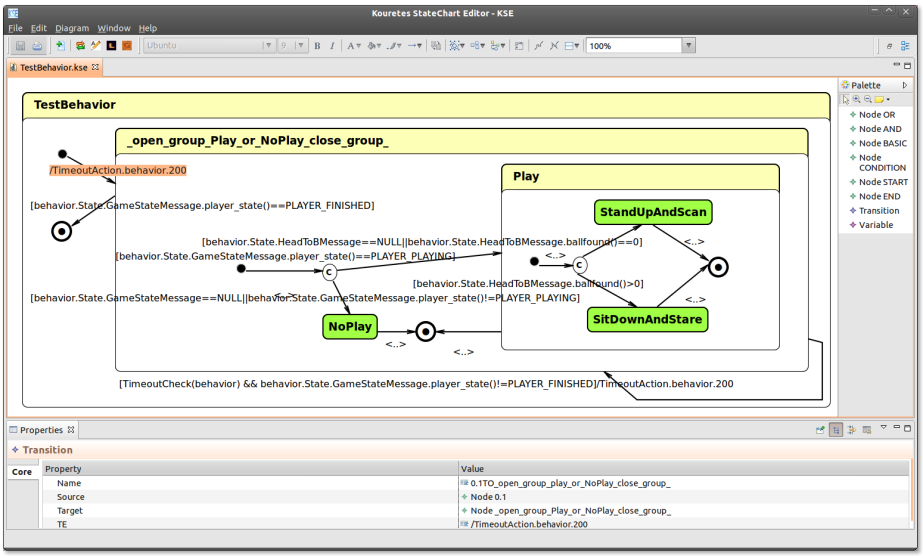

Fig. 4. Kouretes Statechart Editor and the complete statechart for a simple behavior.

References

1. Chatzilaris, E.: Visual-Feature-based Self-Localization for Robotic Soccer. Diploma thesis, Technical University of Crete, Greece (2009)
2. Kyranou, I.: Path Planning for Nao Robots using an Egocentric Polar Occupancy Map. Diploma thesis, Technical University of Crete, Greece (2012)
3. Orfanoudakis, E.: Reliable Object Recognition for the RoboCup Domain. Diploma thesis, Technical University of Crete, Greece (2011)
4. Panakos, A.: Efficient Color Recognition Under Varying Illumination Conditions for Robotic Soccer. Diploma thesis, Technical University of Crete, Greece (2009)
5. Paraschos, A.: Monas: A Flexible Software Architecture for Robotic Agents. Diploma thesis, Technical University of Crete, Greece (2010)
6. Paraschos, A., Spanoudakis, N.I., Lagoudakis, M.G.: Model-driven behavior specification for robotic teams. In: Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems (AAMAS) (2012)
7. 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)
8. Spanoudakis, N.I., Moraitis, P.: Using ASEME methodology for model-driven agent systems development. In: Agent-Oriented Software Engineering XI, Revised Selected Papers of the 11th International Workshop AOSE 2010, Lecture Notes in Computer Science, vol. 6788, pp. 106–127. Springer (2011)
9. Topalidou-Kyniazopoulou, A.: A CASE Tool for Robot-Team Behavior-Control Development. Diploma thesis, Technical University of Crete, Greece (2012)
10. Topalidou-Kyniazopoulou, A., Spanoudakis, N.I., Lagoudakis, M.G.: A CASE tool for robot behavior development. In: Proceedings of the 16th RoboCup International Symposium (2012)
11. Tzanetatou, A.D.: Interleaving of Motion Skills for Humanoid Robots. Diploma thesis, Technical University of Crete, Greece (2012)
12. Vazaios, E.: Narukom: A Distributed, Cross-Platform, Transparent Communication Framework. Diploma thesis, Technical University of Crete, Greece (2010)