Kouretes 2010 SPL Team Description Paper*

E. Chatzilaris, I. Kyranou, E. Orfanoudakis, A. Paraschos, E. Vazaios,
N. Spanoudakis, N. Vlassis, M. G. Lagoudakis

Technical University of Crete (TUC), Chania 73100, Crete, Greece
www.kouretes.gr

1 Team Kouretes (*Κουρήτες*)

Team Kouretes (the only RoboSoccer team in Greece) was founded in 2006 and became active in the Four-Legged league by participating in the technical challenges at RoboCup 2006 and in the soccer games at RoboCup German Open 2007. In 2007, the team began working with the newly-released Microsoft Robotics Studio (MSRS) on the simulated RobuDog robot and participated in the MSRS Simulation Challenge at RoboCup 2007, where it ranked **2nd** bringing the first trophy home. In 2008 the team switched to the newly formed Standard Platform League (SPL) and the Aldebaran Nao robots, working simultaneously on the real robots and on the Webots and MSRS simulators. In RoboCup 2008 the team participated in all tracks of the SPL (Aibo, Nao, Webots, MSRS) and won several distinctions: **3rd** place in Nao, **1st** place in MSRS, and among the **top 8** teams in Webots. In 2009, the team participated in the SPL of RoboCup German Open 2009 and RoboCup 2009. Unfortunately, the team was defeated in the early rounds in both competitions because of serious software problems. The intermediate round game against Zadeat at RoboCup 2009 was lost after a coin toss, following a 0-0 tie during the game and the penalty kicks. On the other hand, the team reached the **6th** place in the RobotStadium Nao competition.

In addition to official RoboCup competitions, the team has participated in various local events and/or exhibitions organizing SPL games with other RoboCup teams. These events include RomeCup 2008 and 2009, Athens Digital Week 2008 and 2009, Festival della Creativita 2008, TechnoTown@Villa Torlonia 2008, Hi-Tech Innovators Partenariat 2007, and the upcoming events Micropolis 2010 and SchoolFest 2010. Team Kouretes is also the organizer of the 1st RoboCup Tournament in Greece hosted by the 6th Hellenic Conference on Artificial Intelligence taking place in May 2010 in Athens, Greece.

Team Kouretes is led by Michail G. Lagoudakis (assistant professor with the Department of Electronic and Computer Engineering) and Nikos Vlassis (assistant professor with the Department of Production Engineering and Management). The members of the team for 2010 (in alphabetical order) are:

1. Eleftherios Chatzilaris, graduate student [Localization, Skill Learning]
2. Iris Kyranou, undergraduate student [Obstacle Avoidance]
3. Emmanouil Orfanoudakis, undergraduate student [Object Recognition]
4. Alexandros Paraschos, undergraduate student [Software Architecture]
5. Nikolaos Spanoudakis, laboratory staff [Behavior Control]
6. Evangelos Vazaios, graduate student [Robot Communication]

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

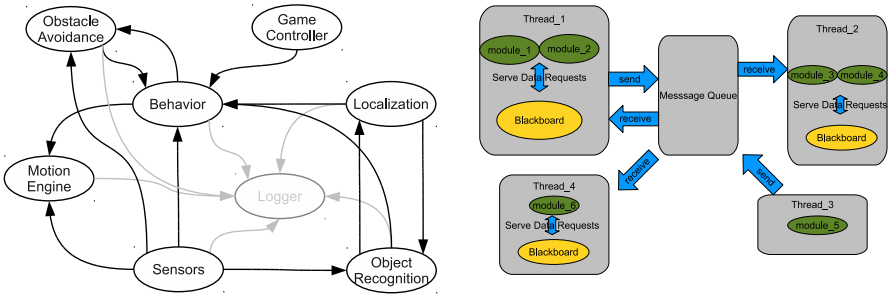


Fig. 1. The Kouretes agent (left) and the Narukom communication framework (right).

2 Team Research

2.1 Software Architecture

Monad is the team's newly developed software architecture whose main goal is to simplify and speedup the cycle of source code development and provide a convenient platform for testing and debugging. Furthermore, Monad integrates already developed technologies and techniques for automation of the building process, platform independence, and concurrent development to relieve the developer from the burden of frustrating tasks (i.e. syncing, NaoQi's version changes).

In the highest level of abstraction, Monad views the robot as a collection of agents, with each agent being responsible for one task on the robot. Each agent decomposes into modules with each module being responsible for a specific functionality. Each module can be described as an operator on data: requires data before its execution, and provides a data output after execution. By using this require-provide model, the architecture is able to create a dynamic schedule of execution inside the agent depending on which modules are currently loaded on the agent and their requirements on frequency of execution. The six basic modules of the single Kouretes agent (Figure 1, left) are:

- **Sensors** Extracting and distributing sensor data from the robot.
- **Object Recognition** Detecting the ball and the goals in the camera frames.
- **Localization** Estimating the position and orientation of the robot.
- **Behavior** Making high-level decisions based on the current perception.
- **Obstacle Avoidance** Planning obstacle-free paths using ultrasonic data.
- **Motion Engine** Executing motion commands requested by the Behavior.
- **Game Controller** Listening to the game controller for the game state.
- **Logger** Logging and storing data from all other modules.

2.2 Robot Communication

During a game, robot players need to share perceptual, strategic, and other team-related information with their team-mates, typically over the wireless network, to coordinate their efforts and achieve a common goal. Towards this end, we have developed a distributed communication framework for robotic teams. Our proposal suggests a distributed and transparent communication framework, called

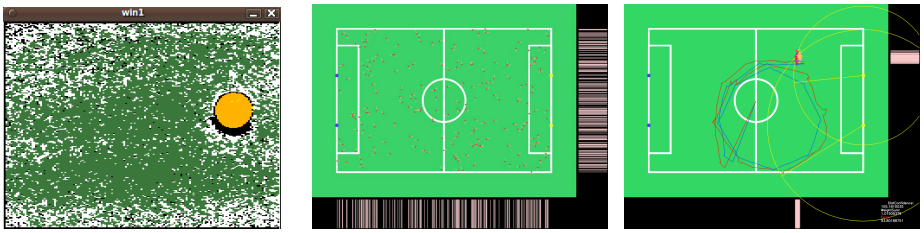


Fig. 2. Vision: dynamic calibration. Localization: true (blue) and estimated (red) path.

Narukom [1], whereby any node (robot or remote computer) can access on demand any data available on some other node of the network in a natural way. The framework 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 the nodes of the team are stored on local blackboards which are transparently accessible from all nodes. Communication is achieved through messages tagged with appropriate topics and relayed through a message queue which is implemented using Google protocol buffers. To address synchronization needs, which are common in robotic teams, we have integrated temporal information into the meta-data of the underlying messages exchanged over the network. Narukom's distributed nature and platform independence make it an ideal base for the development of coordination strategies and for distribution of resource-intensive computations over different nodes.

2.3 Vision

The objects of interest in the SPL are characterized by unique colors. Since artificial light sources play an important role in color recognition, white and gray surfaces are utilized to provide a reference point for the white balance of the surrounding environment and calibrate the camera's white point dynamically. This trick produces balanced camera images and simplifies the problem of color recognition. Our approach is based on labeling by hand a representative set of images from the robot camera and training a classifier which generalizes over the entire color space. The illumination problem is addressed either by including special illuminant features (average color values from a large region or from the entire image) in the classifier, or by transforming the input to an appropriate reference illumination level through histogram specification before classification. These procedures have been integrated into the Kouretes Color Classifier (KC^2) graphical tool [2], which provides intuitive means for labeling images (regions, clusters), selecting features (neighborhood, illuminant), training classifiers (Decision Trees, Support Vector Machines, Neural Networks), and generating code for efficient execution on the robot. The KC^2 tool minimizes the user time required to generate a good color recognizer. Combined with the dynamic camera calibration, a single recognizer yields satisfying results in a wide range of lighting environments. Figure 2 (left) shows an example of using a recognizer trained for a bright environment directly in a dark environment through dynamic calibration. For real-time operation, color classifiers are turned into color tables (precom-

puted lookup tables giving the learned color class for all possible color inputs in constant time), one for each of the two cameras on the robot. These color tables are applied selectively on the YUV images provided by the hardware by a light-weight object recognition scheme which begins with a quick sampling of pixels across the image to locate areas of interesting colors. These areas are then processed locally and examined for validity as field objects. Multiple matches for various objects (e.g. multiple balls) are evaluated and sorted and only one match for each unique object is finally extracted and returned as perceived along with an estimate of its distance and bearing.

2.4 Localization

Self-localization of the robots in the field is accomplished using *KLoc* [3] which realizes Monte Carlo localization with particle filters. The belief of the robot is a probability distribution over the 3-dimensional 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, it is necessary to obtain a motion model for the available high-level actions of the robot (walk, turn, etc.) and a sensor model for its landmark perception capabilities (goal recognition) over the (x, y, θ) space. We constructed a simple motion model $P\big((x', y', \theta')|(x, y, \theta), a\big)$ for predicting the probability distribution of the pose of the moving robot after each action a and a simple sensor model $P\big(z|(x, y, \theta)\big)$ for correcting the belief towards the poses where the probability of obtaining the current observation is higher. The parameters of these models are learned through experimentation on a simulated or real robot. Belief update is performed using an auxiliary (AUX) particle filter:

- temporarily propagate the particles through the motion model
- temporarily weigh each particle using the sensor model and normalize
- resample the original particle population using the temporary weights
- propagate the particles through the motion model
- weigh each particle using the sensor model and normalize

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 2 shows an example of global localization with six landmarks: the left/right blue/yellow goalposts by themselves and the blue/yellow goals as a whole.

2.5 Motion

Effective motion control on the Nao is probably the greatest challenge for all SPL teams. For locomotion, we currently rely on the recently-released proprietary walk engine provided by Aldebaran Robotics, which offers sufficient stability for a variety of surfaces and can even handle small disturbances. Our motion engine allows the independent control of the head, which is typically used for tracking objects of interest, and the lower body, which is used for locomotion.

In addition, it provides a robot safety monitor, which removes the stiffness from the robot joints, as soon as a fall is detected, to protect the robot from damaging itself. It also allows the execution of complex actions (stand-up, kick, etc.) and provides the means for interrupting a motion in progress when it is important to switch to another motion. We use the *Kouretes Motion Editor* (KME) [4, 5] for designing complex actions. The main idea behind KME is the ability to generate, capture, store, manipulate, edit, replay, and export timed sequences of complete robot poses, which resemble the desired complex action. KME provides an abstract motion design environment, which hides away the details of low-level 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. Of particular interest to RoboCup teams is KME's ability to export the symmetric motion (with respect to the sagittal plane of the robot), as well as the temporally reverse motion. In practice, that implies that it is sufficient to design a right kick; the left kick can be automatically exported. Similarly, a stand-up motion, if reversed, will result in a sit-down motion. We have used KME to design a number of complex motions, such as kicks, goalkeeper dives, stand-up routines, and recently complete choreographies. KME v1.0 is freely available through our website, while KME v2.0 is currently being finalized.

2.6 Learning

It is important for our robots to maintain their balance while performing some complex action, such as a kick, that 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 keep 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 slow speed. Our current efforts focus on Monte-Carlo EM-type reinforcement learning methods [6] for optimizing our current controller, aiming at real-time operation at any motion speed and robustness over the entire range of balancing cases.

2.7 Obstacle Avoidance

The SPL rules postulate severe penalties for player pushing, therefore we have dedicated considerable effort in avoiding collisions while moving. Each robot builds and maintains a local obstacle map of its surrounding. This map is stored in a 7×36 polar grid (10cm and 10° resolution, respectively), which covers a 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 two ultrasound sensors located in the chest of the robot. The polar topology facilitates the mapping of the sensor data directly on the map and also captures the resolution of the sensors which

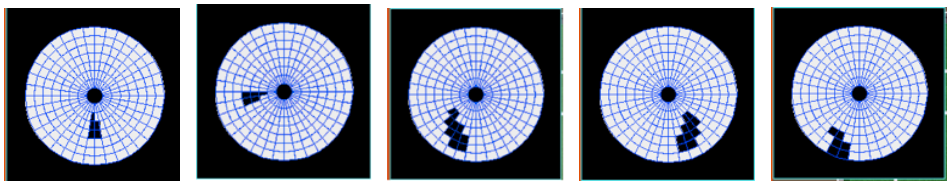


Fig. 3. Transformations of the polar obstacle map: (a) initial map, (b) 80° counter-clockwise rotation, (c) 20cm translation to the right, (d) 20cm translation to the left, (e) translation to the right and counter-clockwise rotation.

is dense close to the robot and sparse away from the robot due to the conical shape of the ultrasonic beams. Locomotion of the robot implies appropriate transformations on the polar grid. Rotational motions of the robot can be easily translated into rotations of the polar map, however translational motions of the robot require appropriate translations of the polar map through geometric transformations. To avoid costly computations for these transformations we have precomputed and stored these transformations for a discretized range of possible translational motions and therefore the polar map can be updated in constant time, nevertheless with some loss of accuracy. Examples of map transformations for various motions are shown in Figure 3.

Each cell in the polar maps stores the probability that there is an obstacle at the corresponding position in the field. The values of all cells are initialized to 0.5 to indicate the absence of any information. Given a reading from an ultrasound sensor, the values of all cells within the sensor cone are updated according to a simple sensor model that increases the value at the distance of the reading (obstacle) and reduces the value at all intermediate cells (free space). Since most obstacles in the SPL field are dynamic (moving players), an aging update step gradually brings the values of all cells towards 0.5 (full uncertainty) over time, unless they are updated due to sensor data. The polar obstacle map can be used to plan obstacle-free paths for the robot in any desired direction of motion or to any desired destination. An obstacle-free shortest path to the center of the map (the current robot position) is derived from the cell on the periphery of the map that matches the desired direction or from the map cell that corresponds to the desired destination. This path indicates how the robot should move to reach the desired direction or destination quickly while safely avoiding all mapped obstacles. Paths can be computed through A* search for single destinations or through value iteration for multiple destinations (of which one is to be chosen). Examples of path planning and obstacle avoidance are shown in Figure 4.

2.8 Behavior

Since RoboCup players are agents, we have selected the Agent Systems Engineering Methodology (ASEME) [`http://www.amcl.tuc.gr/aseme`] methodology for modeling the behavior of the Kouretes team. ASEME allows for modular development, so that specialized teams can develop different modules and use the Intra-Agent Control (IAC) model to glue them together. IAC is defined by the Agent Modeling Language (AMOLA) as a statechart [7], which allows for mod-

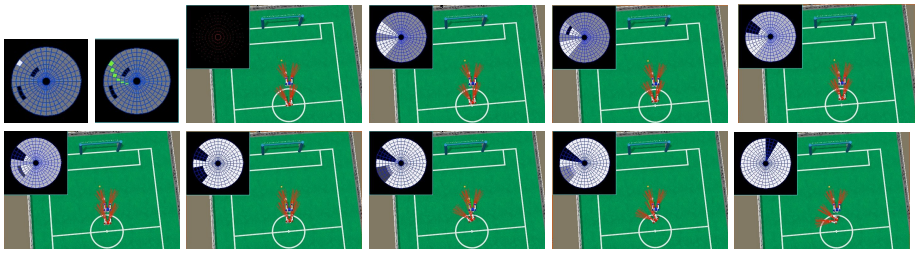
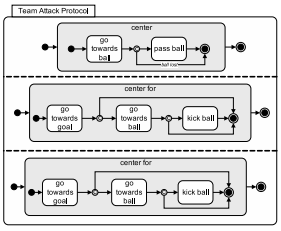


Fig. 4. Path planning on the polar obstacle map and an example of obstacle avoidance.

eling both the functional and dynamic aspects of a system incorporating the activities that are executed and their control information. Statecharts can be rather complex, therefore ASEME uses the System Roles Model (SRM) in the analysis phase to indicate the process that the agent will follow with the liveness formulas that are easier to conceive and that can be automatically transformed to the IAC model using Model Driven Engineering (MDE) techniques. The challenges that we faced in adopting ASEME and AMOLA were four: (a) develop a statechart engine that allows for multithreading as the robots must do a lot of things simultaneously, (b) define the IAC's transformation to executable code for the Nao platform providing for specific laces where the programmers will invoke the needed functionalities, (c) define the message-exchange mechanism and integrate it in the automatically generated code, and (d) define the allowed expressions for the statechart transition expressions.

The following example demonstrates the ASEME development process. Let us assume that in the analysis phase we edit an attack protocol using the Agent Interaction Protocol of AMOLA (figure below, left). The process field is defined as a liveness formula, where we use the Gaia operators [8, 9] for defining the dynamic aspect of the agent system, what happens, and when it happens. Briefly, $A.B$ means that activity B is executed after activity A, A^{ω} means that A is executed forever (when it finishes it restarts), $A|B$ means that either A or B is executed and $A||B$ means A is executed in parallel with B. These formulas can be automatically transformed to the Inter-Agent Control (EAC) model (a statechart) using the ASEME `srm2iac` transformation tool (figure below, right).

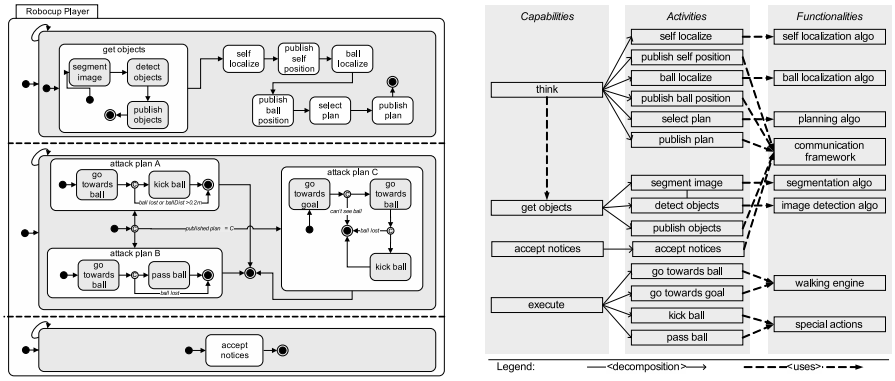
```
Attack Protocol
Prerequisite: Center is closest to the ball
Outcome: Center passes the ball to one of
         the two Center-Fors to make the kick
Process for three participating roles:
   attack_protocol = center || center_for || center_for
   center = go_towards_ball.[pass_ball]
   center_for = go_towards_goal.[go_towards_ball.[kick_ball]]
```



In the EAC the modeler can define transition expressions with events and conditions that will allow for state transitions. Moreover, the EAC shows what a team of robots can simultaneously achieve (in a multi-agent level of abstraction). Then the analyst can focus on the agent level and define individual agent roles implementing parts of the defined protocol. Consider a player role:

```
player = think^ω || execute^ω || accept_notices^ω
think = get_objects.self_localize.publish_self_position.ball_localize.
                                                    publish_ball_position.select_plan.publish_plan
get_objects = get_image_from_camera.segment_image.detect_objects.publish_objects
execute = (go_towards_ball.[kick_ball]) | (go_towards_ball.[pass_ball]) |
                                                    (go_towards_goal.[go_towards_ball.[kick_ball]])
```

Again, using the `srm2iac` tool the developer obtains the IAC model directly from the SRM model and for the parts that are imported from the previously defined EAC the transition expressions are copied as they are. In the figure below, one can see the IAC model for the player role (left) with the capability to participate in the attack protocol either as a center-for or as a center and the functionality table (right) which shows the relationship between capabilities, activities, and functionalities. The last step is to automatically generate the code that controls the behavior of the agent and insert tags where the programmers need to connect their code to the implemented functionality.



References

1. Vazaios, E.: Narukom: A distributed, cross-platform, transparent communication framework. Diploma thesis, Technical University of Crete, Chania, Greece (2009)
2. Panakos, A.: Efficient color recognition under varying illumination conditions for robotic soccer. Diploma thesis, Technical University of Crete, Chania, Greece (2009)
3. Chatzilaris, E.: Visual-feature-based self-localization for robotic soccer. Diploma thesis, Technical University of Crete, Chania, Greece (2010)
4. 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)
5. Pierris, G.F.: Soccer skills for humanoid robots. Diploma thesis, Technical University of Crete, Chania, Greece (2009)
6. Vlassis, N., Toussaint, M., Kontes, G., Piperidis, S.: Learning model-free robot control by a monte carlo em algorithm. Autonomous Robots **27**(2) (2009) 123–130
7. Harel, D., Naamad, A.: The STATEMATE semantics of statecharts. ACM Transactions on Software Engineering Methodologies **5**(4) (1996) 293–333
8. Spanoudakis, N., Moraitis, P.: The agent modeling language (AMOLA). In: Proceedings of the 13th Intl Conference on Artificial Intelligence (AIMSA). (2008) 32–44
9. Wooldridge, M., Jennings, N.R., Kinny, D.: The gaia methodology for agent-oriented analysis and design. Autonomous agents & multi-agent systems **3**(3) (2000) 285–312