The RESCUE Project
Cooperative Navigation for Rescue Robots

Pedro Lima, Luis Custódio, M. Isabel Ribeiro, José Santos-Victor
Instituto de Sistemas e Robótica, Instituto Superior Técnico — Torre Norte
Av. Rovisco Pais, 1; 1049-001 Lisboa; PORTUGAL
E-mail: {pal,lmmc,mir,jasv}@isr.ist.utl.pt

Abstract— **This paper provides an overview of the RESCUE project, a joint work of three research groups at ISR/IST, Lisboa. The main goal of the project is to provide integrated solutions for the design of teams of cooperative robots operating in outdoors environments, with special focus in the short and mid-terms on perception and representation issues, as well as cooperative navigation, and, in the mid to long-terms, on task modeling, planning and coordination as well. Currently, the reference scenario for the project refers to a long-term goal of developing robotic teams to help humans in search and rescue missions, and is based on one land and one aerial (blimp) robot. The scientific advances made on the enabling disciplines during the project lifetime should be extendable to other outdoors applications. A brief description of the on-going research and the results obtained is also provided.**

Keywords— **Topological Navigation, Cooperative Navigation, Task Coordination, Software and Functional Architectures, Rescue Robots.**

I. Motivation

Outdoors environments offer several challenges for service robot applications, especially under scenarios that are dangerous (e.g., search and rescue, fire fighting, demining), long-lasting information-massive (e.g., environment surveillance and monitoring) and/or production-massive (e.g., agricultural and harvesting), therefore too difficult to handle uniquely by humans.

One potential scenario for outdoors robot operation is the search and rescue for victims after a catastrophic event. This may range from natural occurrences, such as earthquakes or floods, to urban riots or terrorist attacks.

Earthquakes are specially daunting phenomena. Even though their occurrence is fortunately sparse, the consequences are often tragic, leading to thousands of deaths and many more injured people, besides mass building destruction. Reports from the infamous 1995 Kobe earthquake in Japan show that many unexpected events hit the communications and civil protection infrastructure, disabling the execution of most of the available plans for disaster situations [6]. Especially the serious damage to the telecommunications network and the fact that the buildings of disaster mitigation organizations were hit, caused serious delays in the arrival of local and external assistance to the victims. Structures in risk of collapsing are often inaccessible to humans because they are too dangerous. Survivors may be unintentionally injured during removal of debris due to the unawareness of their location or presence by the rescue teams.

Teams of heterogeneous robots can represent an invaluable help for future search and rescue operations. Robots can crawl over the collapsed structures, depositing small-sized robots, and feeding air, water, food and medication to trapped individuals through tubes snaked into the collapsed structure. Small-sized robots can sneak inside very confined spaces, taking with them tiny cameras and other sensors to detect survivors and map survivor locations. Aerial robots can provide a broad view of the search and rescue scenario and map high-destruction locations. Cooperation among human-operated stations, a distributed network of sensors located around the disaster area and teams of tele-operated and/or autonomous robots can increase the amount of available information to the rescue teams. Whenever communications with the networked sensors or the sensors themselves fail due to the earthquake impact, robots can be dispatched to cover the areas most inaccessible to humans and help finding victims at those locations. These mobile robot networks can also provide a dynamic view of the scene at relevant locations.

A multi-disciplinary joint venture is definitely required to face such a challenge, so as to address under an integrated framework issues such as scenario mapping, multi-robot (cooperative) navigation, (multi-robot) task planning and coordination. Three research groups (Intelligent Systems, Computer Vision and Mobile Robotics) of the *Instituto de Sistemas e Robótica* at *Instituto Superior Técnico* (ISR/IST), Lisboa, have joined their research efforts on outdoors search and rescue robots under the Portuguese funded project "RESCUE – Cooperative Navigation for Rescue Robots".

The main goal of the project is to provide integrated solutions for the design of teams of cooperative robots operating in outdoors environments, with special focus in the short and mid-terms on perception and representation issues, as well as cooperative navigation, and, in the mid to long-terms, on task modeling, planning and coordination as well. Currently, the reference scenario for the project refers to a long-term goal of developing robotic teams to help humans in search and rescue missions, and is based on one land and one aerial (blimp) robot. Simplifying assumptions include daylight operation and good meteorological conditions (weak winds and no rain). The project

This work is supported by project PCTI / 1999 / SRI / 33293 and ISR/IST Pluriannual funding from the *Fundação para a Ciência e a Tecnologia*, as well as by POSI, in the frame of the QCA.

also concentrates on search and mapping operations, rather than on the actual rescue, since this would require a focus on technology that is out of its current scope.

The scientific advances made on the enabling disciplines (Computer Vision, Robot Navigation, Hybrid/Discrete Event Systems and Artificial Intelligence) during the project lifetime should be extendable to other outdoors applications, such as environmental monitoring and surveillance, satellite formations or planetary exploration, where the group plans to be involved in the future.

Simultaneously, some of the project members have engaged in an industrial consortium project with a Portuguese SME, whose goal is to build the prototype of a semi-autonomous robot to be used in real search and rescue applications. The project has a planned duration of two and a half years, after which the robot will undergo tests at destruction scenario used by the Lisboa Fire Department for practical training of human fire fighters. The semi-autonomous robot will be designed and developed in continuous interaction with the Lisboa Fire Department experts and will be endowed with an on-board computer and temperature-sensitive infrared camera, color image camera, sonars, inertial sensors, temperature, gas and humidity sensors. The two projects can clearly benefit from each other. The research-oriented project will constrain some of its options based on the suggestions from the Fire Department experts on what are realistic scenarios. The development-oriented project will benefit from the research advances to create new technology transferable to real scenarios.

II. Reference Scenario and Research Summary

A reference scenario for the RESCUE project was set up with two main goals: it should refer to a reasonably realistic situation, and it should be rich enough to accommodate all the research topics of interest for the involved groups. The current scenario consists of two robots: one aerial blimp/zeppelin and one land outdoors robot. The aerial robot will perform several tasks, such as making a vision-based topological map of the destroyed site. The map will include information on the relevance of each of the mapped locations concerning the degree of destruction, presence of victims, etc, as well as on the difficulty of traversing regions between them, due to the presence of debris or obstructed paths. The map will be stored as a graph and will be used to choose the best path for the land robot to reach a goal location (e.g., one with a larger number of victims). It can also be used to help the aerial robot navigating. Issues of representation arise here, as the views of the land and aerial robots are different, even when they refer to the region associated to the same node of the topological graph. The land robot will use several sensors (GPS, inertial, vision, laser scanner, sonars) to navigate towards the goal, handling the details associated to the path (e.g., debris, trees, people on the way, etc). This will extend the scope of the research to metric navigation. Nevertheless, the topological map obtained by the aerial robot can also be used as the initial iteration of a topological mapping algorithm based on cooperative information from the aerial and land robots.

While the land robot moves towards the goal location, the aerial robot should follow it using a formation control algorithm, so as to keep a reliable communication link and to serve as a relay for information that the land robot may need to send to distant stations. Therefore, formation feasibility [12] and control is also covered, including situations of temporary occlusion (from the communications or vision standpoints) of one of the vehicles.

An animation illustrating the scenario above was developed to better explain the reference scenario to project members and non-members (check http://rescue.isr.ist.utl.pt/videos/rescue_web.mpeg).

Considering the long term goals of the project, this and future scenario extensions (e.g., by adding more heterogeneous robots) require a *software architecture* capable of handling real-time distributed control and supervision systems, as well as a *functional architecture* to integrate all the required subsystems (and their corresponding functions) consistently, i.e., so as the whole system is designed from performance specifications.

The software architecture, currently in its implementation stage, is based on a multi-thread and distributed blackboard approach. The different threads handle sensor data acquisition and processing, communications, behavior execution and behavior switching. Raw and processed data go to the distributed blackboard, a distributed shared memory organized in classes of variables corresponding to the different sensor classes. Some variable values are kept locally at the robot that acquired and processed the data. Others are broadcasted (or sent individually) to the teammates. Some transducers have an associated set of virtual sensors, each of them implemented by a separate *agent*. A typical example is the vision transducer, with associated sensors such as locate object, determine object speed, classify object.

The functional architecture comprehends the following subsystems: task planning (to handle task/behavior allocation, intra-team communications and formation topology, when a formation is required), task coordination (to handle the decisions on which behaviors to select at every time step, so as to optimize the performance in terms of time taken, reliability, cost, utility or other factors - requiring performance feedback from each subsystem) and behaviors (individual – referring to one single agent – and relational – involving relations among more than one teammate). Behaviors can be built as finite state automata, where arcs are associated to conditions over detected events and most states correspond to dynamic control systems implementing navigation functions, parameterized according to the task at hand (e.g., move until a given posture is reached, track an object or a teammate). Note that behaviors can be implemented using alternative approaches (e.g., based on Petri nets or production rules), with relative advantages concerning the representational power but a potentially reduced analysis power. This will be an issue subject to research under the project. However, the software architecture is open enough to host these alternative models.

The navigation functions are the key practical develop-

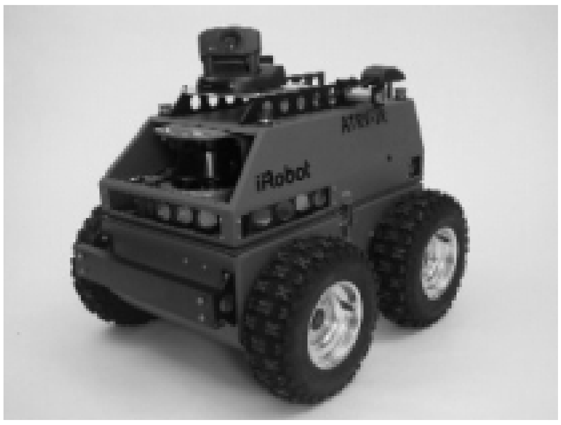


Fig. 1. The ATRV-Jr land robot, fully equipped.

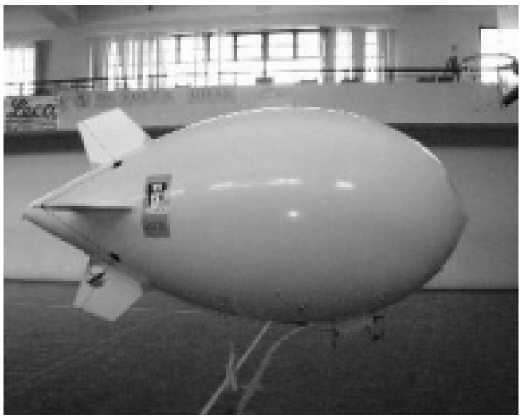


Fig. 2. The blimp aerial robot, fully equipped.

ments expected during this project. They will be based on a multi-sensor system (GPS, compass, odometry, laser scanner, sonars, vision and inertial navigation system composed of rate-gyros and accelerometers) for the land robot, an ATRV-Jr, shown fully equipped in Fig. 1. The navigation of the blimp (shown in Fig. 2) will be mostly vision-based. Both the blimp and the ATRV-Jr navigation systems will mix metric and topological navigation.

The current work concerning metric navigation under the project builds on existing closed loop solutions which use: the GPS as a reference and odometry plus accelerometers as feedback sensors for **position** estimates; the compass as a reference and the rate-gyros as feedback sensors for **orientation** estimates (roll, pitch and yaw). The novelty here is on handling particular problems of outdoor environments, such as very poor odometry and frequent loss of GPS information due to trees or buildings. The quality of metric navigation is improved by cooperative navigation, where a network of communicating sensors assembled on the team robots can take advantage of their space diversity to provide information to each of the team robots which they might miss if operating alone (e.g., one of the robots does not have GPS available but one of its teammates does and can also estimate their relative postures). All this work will necessarily require sensor models, which cannot be obtained before outdoor runs, during which data from all navigation sensors is registered, are performed. The recorded data can subsequently be subject to statistical analysis in order to build the sensor models. This work is currently underway but it is not reported here.

Work concerning topological navigation and map building shares two different, complementary, approaches. The first, described in this paper in Section III, builds a topological map based on features extracted from the environment using all the on-board sensors and localizes the vwhicle within the topological map. This approach is complemented by vision-based topological navigation, which is rooted on existing approaches to traverse a region by identifying successive images representing a mosaic of sub-regions previously obtained. In the latter, the research work will concentrate on low complexity representations of the mosaic images that can still allow the determination of where the vehicle is with a given accuracy (typically lower than for metric navigation [3]. Its novelty comes also from the fact that the mosaic is constructed by the navigating robot itself or by one of its teammates (e.g., the blimp builds a topological map for the ATRV-Jr navigation). The sharing of topological maps among teammates with very different postures and world views will also raise challenging representation research issues and provide improved individual navigation capabilities.

The project approach is based on an integrated view of the overall system to be developed in the short and long terms. As such, a relevant issue concerns sensor data fusion and the availability of (raw or processed, including fused) data in a distributed blackboard [8]. Information on features relevant for the system is obtained in general from different sensors, by fusing them according to sensor models (e.g., considering their measurement uncertainty) and stored in the blackboard. The blackboard provides a sensor database where all the relevant raw and processed data (e.g., from a given time period) is organized and available to all decision levels, as well as information shared among teammates, including data not acquired and/or processed by the team member using it. This distinguishes our approach from non-integrated ones, where data is acquired directly by each subsystem from the sensors relevant for that subsystem, disregarding the fact that this information may be relevant for other subsystems. A typical example is the information on the presence of one or more obstacles near a robot. This information is immediately required for the robot to avoid the obstacle(s) but it is also relevant to update the team world model, needed for decisions at the task planning subsystem.

III. On-Going Research and Preliminary Results

In this section we briefly report on-going work and preliminary results on the topological navigation of the land robot (shown in Fig. 1), the vision-based control and guidance functions library for the aerial robot (shown in Fig. 2) and the design of the software and functional architectures for the whole system.

A. Land Robot Topological Navigation

A.1 Map building

A topological map supports the navigation of the land robot. To perform a symbolic representation of the environment the robot perceives it with its on-board sensors and the acquired data is processed aiming at extracting the most relevant features of the environment. The robot perception is condensed in observations, o_t, that represent the information obtained from the processing of the raw data acquired at each time instant t. An observation is a vector where each component relates to a different feature. For instance, a feature defined as Color might have the values *red*, *green* or *blue*. The different components of the observations reflect that the robot is able to perceive special types of attributes of the environment. These different levels of perception have to be recorded in the map, that is composed by a set S of N states s_i, $i = \ldots, N$. Each state, containing a partial representation of a physical area of the environment, is characterized by a set of relevant features to support the state identification and to avoid mismatching. According to the uncertainty of the measurements and sensors, each state s_i is represented by a Gaussian pdf.

With this map characterization, the mapping procedure consists of estimating, for each state in the map, the mean vectors and the covariance matrices of the Gaussian pdfs that maximize the probability of all observations given the environment model, i.e., that maximize the likelihood function or, equivalently, its logarithmic representation,

$$L(S) = \log(p(O \mid S)) = \sum_{i=1}^{t} \log\left(\sum_{k=1}^{N} c_k \cdot p(o_i \mid s_k)\right) \quad (1)$$

where any observation o_i is a measurement of the state s_k with a weight c_k ($\sum_{i=1}^{N} c_k = 1$), which depends on the number of observations characterizing that state. The larger the number of observations, the larger c_k will be [17]. $L(S)$ can be seen as the likelihood of the environment mapping by the N states, based on the observations until time t.

The maximization of the likelihood function in (1) is a hard problem to solve. A way to overcome its computational burden is by changing the function $L(S)$ by the expectation of the likelihood given a previous estimation of the model, this corresponding to the use of the Estimation and Maximization (EM) algorithm, as referred in [13], [15] and [10].

The number of states, which is constant during the EM algorithm, does not necessarily guarantees an accurate environment representation. Even for a good representation at a given time instant, as the robot is always acquiring new measurements, a possible update of the number of states might be required. Consequently, it is strictly necessary to evaluate the number of states after the stabilization of the EM algorithm, as described with more detail in [16].

A.2 Localization

The robot estimated location $\hat{q}_t$ is the map's state that is most likely to have produced the observations acquired by the robot sensors during a given time interval. This is notably different from the usual localization procedures that aims at providing an accurate pose (position and orientation) estimation in a local or global frame. In fact, when the proposed localization procedure yields a robot estimated location, $\hat{q}_t = s_i$, this does not mean that the robot physical pose coincides with the actual location that lead to the map state s_i through its several observations by the robot.

The state estimation at each time instant t is evaluated using all the available observations during the interval T. According to a probabilistic approach, the current state estimation, $\hat{q}_t$, is the argument that maximizes the pdf of the state given the observation sequence $O_T = \{o_1, o_2, \ldots, o_t, \ldots, o_T\}$ acquired in the time interval T, i.e.,

$$\hat{q}_t = \arg\max_{q_t} P(q_t = s_i \mid o_1, \ldots, o_T) \quad (2)$$

Based on Markov Models, the localization procedure in (2) is similar to the high-dimensional maximum likelihood estimation problem. This problem is efficiently solved using the Baum-Welch algorithm, as well as the Forward-Backward (FB) algorithm or simply the Alpha-Beta algorithm. The same problem is referred in [14], [5] and in [11] as a special version of EM.

In the FB algorithm, the time interval has a fixed length. For long time intervals, corresponding to large operating periods, the FB algorithm implementation becomes too time consuming. For this reason, it is necessary to understand what has to be changed in the algorithm if the length of the observation sequence is also considered to be a variable. The revisited FB algorithm is described in [17].

A.3 Results

The map building algorithm was tested by tele-operating the robot from a corridor to a room, containing chairs, tables and people moving. Setting low accuracy to the mapping algorithm the result is a topological map as depicted in Figure 3. The map contains two states, corresponding to the room and the corridor, which were distinguished based on the differences detected on the free-area, as the selected feature. The states do not contain any parametric information and the connections between them are described in [17]. The Figure 3 depicts the measurements of the laser and sonar with two different grey levels, corresponding to each state.

B. Aerial Robot Vision-Based Control

The blimp is a non-holonomic robot with three actuators:

- two parallel non-independent DC motors (with propellers) mounted on a shaft located at the bottom of the blimp through the gondola. Since the shaft can rotate of 360°, the complete device can control motion in the upwards/downwards and forward/backward directions.

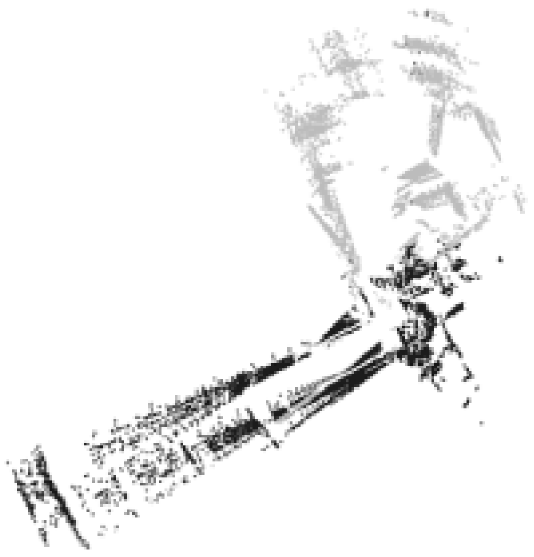
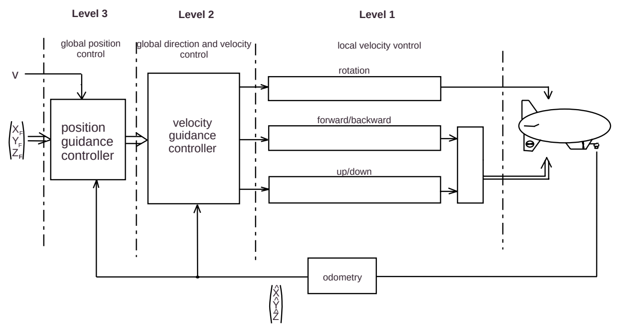


Fig. 3. Topological map with two states: a corridor and a room


Level 3
Level 2
Level 1
global position control
global direction and velocity control
local velocity vontrol
V
$\begin{pmatrix}X_F\\Y_F\\Z_F\end{pmatrix}$
position guidance controller
velocity guidance controller
rotation
forward/backward
up/down
odometry
$\begin{pmatrix}\hat{X}\\\hat{Y}\\\hat{Z}\end{pmatrix}$
Fig. 4. Blimp guidance control block diagram.

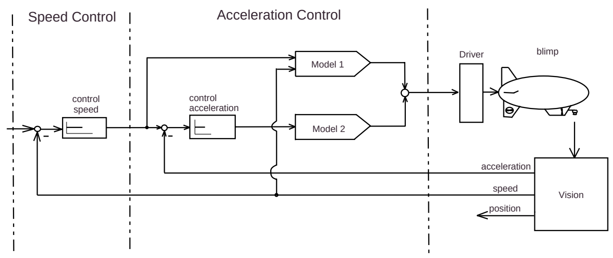


Fig. 5. Low level blimp control block diagram.

• a third DC motor (with propeller) is located at the tail in the lower wing to enable rotation.

The only available sensor is a small camera with a radio link to a ground computer. This computer also has a radio link — a modified remote controller — to send signals to the three actuators of the blimp (the gondola servos for the propellers and shaft rotation, and the tail servo).

A motion control library has been developed with the goal of achieving desired positions (x, y, z) in world coordinates. In general, to control the position, a multi-variable nonlinear dynamic controller, robust to considerable (e.g., moderate winds) disturbances is required. In order to design such a controller, a model of the blimp dynamics is needed. These tasks are simplified, at the cost of possible reduced robustness, by using feedback linearization to cancel non-linear terms at the guidance and lower control levels. This was the approach followed here.

The guidance controller, depicted in Fig. 4, is split in its position and velocity controllers (levels 3 and 2 in the figure, respectively). It uses estimates of the current position provided by visual odometry to lead the blimp to the desired final posture (x_F, y_F, z_F) with a velocity v by generating references to the blimp velocity control loops for rotation, forward/backward and upwards/downwards degrees of freedom (dof – level 1). At levels 2 and 3, world coordinates are used, while local robot coordinates are used at level 1.

The guidance controller will be used for several purposes, such as moving the blimp to a target posture, following lines on the ground (e.g, a road) or tracking a moving object (e.g., the land robot).

The lower level controllers for the controlled dof are depicted in Fig. 5. Speed estimates are obtained from optical flow [18]. Actually, the 6 dof in configuration space can be determined using this method, if needed. Acceleration is determined for each loop from the derivative of speed.

All three low level controllers are similar. So we will present one of them in more detail.

The forward/backward controller block diagram is shown in Fig. 5. It is based on an outer loop composed of two cascaded feedback control loops for speed and acceleration, and an inner loop to predict the required force and cancel the model non-linearities. The inner loop controller reduces the errors due to perturbations such as wind and unmodeled dynamics in the outer loop.

The simplified dynamic model of the blimp is given by

$$m\dot{v} = F + \rho v^2 \tag{3}$$

where F is the applied force, resulting from appropriate servomotor torques, m is the blimp mass and ρ is a coefficient that depends on the blimp geometry. The equation for Model 1, used in the inner loop shown in Figure 5, is

$$F(\dot{v_d}, \hat{v}) = m\dot{v_d} + \rho\hat{v}^2 \tag{4}$$

where $\dot{v_d}$ is the desired acceleration and $\hat{v}$ the current estimate of the blimp velocity. The ρ coefficient is determined out of a simple experiment. With zero acceleration, Equation 3 becomes $F_{ss} = \rho v^2$, or

$$\rho = \frac{F_{ss}}{v^2}. \tag{5}$$

Therefore, one can determine experimentally the value of ρ by providing a know input force and measure the steady state speed ($\dot{v} = 0$).

Regarding Model 2 in Figure 5, the equation is

$$F(\ddot{e}) = m\ddot{e}. \tag{6}$$

where $\ddot{e}$ is proportional to the difference between the desired and estimated accelerations. As for the controllers, they are implemented by constant gains determined from the power required for the maximum acceptable error.

C. Software and Functional Architectures

The software architecture was designed to provide an integrated set of operations, namely graphical task design, task planning, task execution, task coordination and task analysis for a multi-agent and multi-robot system. The architecture must support fusion of information acquired by several sensors and sharing information between the robots by a blackboard, therefore it is geared for the cooperation between robots [8]. Its main goal is to close the gap between hybrid systems and software agent architectures, building on the experience from earlier tools [1], [4], [7], [9].

The main building blocks of the RESCUE project software architecture are *agents*. Different types of agents are combined hierarchically in a distributed arrangement [2]. The backbone of the architecture is an agent hierarchy. At the top of the hierarchy, the algorithms associated with the agents are likely to be planners, whilst at bottom they will be interfaces to control and sensing hardware. The planner agents are able to control the execution of the lower level agents to service high-level goals. To offer platform independence, only the lowest level agents should be specific to any hardware, and these should have a consistent interface for communication with the planning agents that control their execution.

In the next subsections, we describe the novel features of our software architecture, including its three basic components, as well as the five main **Execution Modes** for each of those components. The components are the *Agents*, the *Blackboard*, and the *Control/Communication Interface*. The *Control Mode* coordinates the run-time interactions between the basic components. The *Design, Calibration, Supervisory Control* and *Logging and Data* Modes concern the programmer interface. All modes are summarized in the sequel.

C.1 Agents, Blackboard and Control/Communication interface

We define an Agent as a software element with its own execution context, its own state and memory and a built-in procedure to sense and take actions over the environment. Each of the agents has two interfaces: one to the upper-level agents, the other to the lower-level agents, shown schematically in Fig. 6.

An agent is implemented as an active object with two defined ports in the upper interface. One of the ports is the **input port**, which can be seen as the *sensing port*, through which the agent is notified of changes in the world. In can also be seen as the *request port* through which the agent receives from higher-level agents notifications of actions it should perform. The other port is the **output port**, through which the agent reports progress to the caller or throws events to higher-level agents. This is what we define as the consistent interface for communication and control.

An agent has also a lower level interface through which it can control and sense its lower level agents. The lower level interface is customized according to the agent type, e.g., a finite state automata agent has as many lower level control ports as agents it is controlling and one lower level input port where all lower level agents write events.

Agent's ports are linked together through the blackboard. To provide a flexible agent hierarchy, the agent upper ports are never assigned in the definition of the agent, but rather in the task definition. Ports are actually a synchronized data entry in the blackboard. Each agent defines a new scope (its scope) inside the blackboard. This scope can be viewed as the agent memory.

Different types of agents are supported:
Goal-Based Agent: an agent that knows the other agents' actions, the context where to apply its actions and the expected result of those actions, so as to build a plan to reach its goal;
Agent Driven by a Table with an Internal State: i.e., a Finite State Automata;
Concurrent Agent: an agent that represents the concurrent execution of two or more agents;
Control Loop: an agent that executes a closed loop controller, or an sense-think-act loop, at a constant rate;
Cooperative Agent: an agent that represents the state of the interaction between two or more agents;
Sensor Fusion Agent: an agent that collects data from several sources, and fuses that data for later use by some other agent;
Sensor Agent: a driver or a sensing hardware device server;
Actuator Agent: a driver or an actuator hardware device server.

Combinations between these agent types provide the flexibility to build a functional architecture suited for a given multi-robot (cooperative) population. For special interactions that are not supported, the architecture is open, so as to include other types of agents.

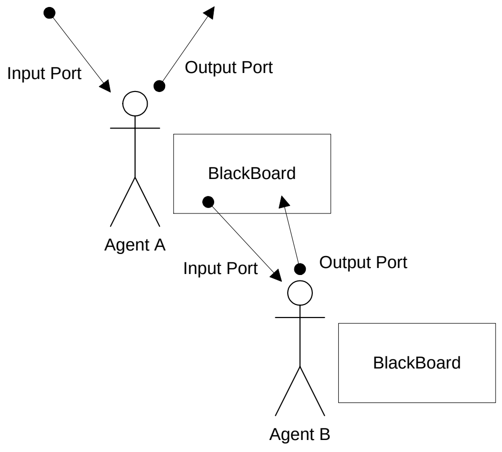


Fig. 6. Schematic representation of an agent upper and lower interfaces and their input and output ports, as well as their connection with the blackboard.

Control Mode: An upper level agent can send commands to a lower level agent through special and well-defined functions: start, stop, set and reset. If the agent that encompasses the whole agent fleet is stopped, it will request its hierarchically dependent agents to stop, triggering a cascade reaction that will stop all the lower level agents under its command. A similar description applies to the other commands.

Design Mode: The Design Mode is similar to a Graphics Drawing Program. Such programs provide different tools for the different graphic objects, e.g., lines, squares. Under the Design Mode, instead of drawing tools for each type of graphic we have a drawing toolbox for each type of supported agents, plus another one to link agents written from pure code. The output is a meta-language that represents an instantiation of the supported agents or the included code files when the agent is built in pure code. This meta-language is then transferred to the target robots for execution [4].

Calibration Mode: To facilitate the calibration procedure of robot fleet, each agent has a calibration window, which can be requested remotely before the start of the mission or during its execution. The calibration data is persistent and can be used in a later mission. That data is stored in the central station, making fleet management easier. Calibration data is distributed to the robots before run time. The operator makes the calibration following the instructions appearing in the remote window. For each agent which is part of the mission to be executed, the operator is asked whether he/she wishes to make a new calibration, or to skip, save or load a previous one. This is done top-down. Answering skip to the agent that represents the whole fleet will calibrate all robots using the most recent calibration data. This framework provides support to data management calibration files. It also supports the tools to write and read the various data types to and from files.

Supervisory Control Mode: Each of the agents has, in addition to the Calibration Window, a Supervisory Control Window that is designed to be user-friendly. Therefore, the agent that controls the motors has an interface that should be appropriated for doing that. This interface should then be different from the interface of a planner agent. There are common features to all agents, such as the requests to start, stop or logging. All the common features are provided by a software tool, in the form of buttons and text boxes. The supervisory control window uses the same program interface, through which the agents receive control requests from higher level agents and data through which the lower level agents report success, failure or progress to the higher agents. The only difference is the use of a graphical window to interface humans. If the user chooses to control an agent which is part of the hierarchy, the tool should disable all control requests arriving at the controlled agent from other agents. In the supervisory control window, there is also a blackboard view. In the blackboard view, the human supervisor can consult or modify the various types of variables.

Logging and Data Mode: Each agent can keep a logging file. If the supervisor chooses an agent to make data log files, such a file is written locally, i.e., inside each of the robots. After finishing its mission, the logging files are stored on the central station. During run-time, an operator can also choose to consult the logging of a particular agent. The framework with the corresponding time tag logs all the requests arriving and all reports departing from an agent. Changes in the blackboard variables can also be chosen to be automatically logged. Additional logging should be done inside the code of the agent. The framework will provide a program interface for doing so, without the need to open files, managing files and so.

C.2 Functional Architecture

The combinations between available agents provide the flexibility to build the functional architecture for the Rescue project. Let us show that with a relatively simple example related to the project reference scenario: consider that the land robot includes an agent that builds a topological map, another agent that determines the robot localization and a third agent that handles the navigation loop. The map and the robot pose are shared by the three agents through a blackboard shared memory inside the robot agent (an agent at a hierarchically higher level). The four agents are running in parallel, so a **Concurrent Agent** is required within each of the agents (see Fig. 7).

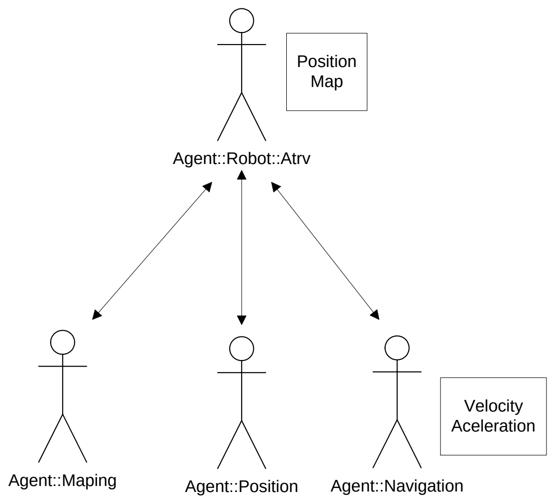


Fig. 7. The robot agent for a robot endowed with Mapping, Positioning and Navigation agents.

Let us now assume that the navigation agent has available alternative behaviors, represented by alternative agents (see Fig. 8). When the robot starts its motion, it does not have a map yet. Hence, the navigation agent should wander around for a while and then receive commands to move from one place to another. As a result, we further decompose the navigation agent into two lower level agents: one to wander around (Wander), another for path following (PathFollow). In a scenario under which the land robot needs to follow the aerial robot, a third agent is needed (FollowTarget). One might think of a Finite State Machine connecting the PathFollow, Wander and FollowTarget agents (representing land robot behaviors) as

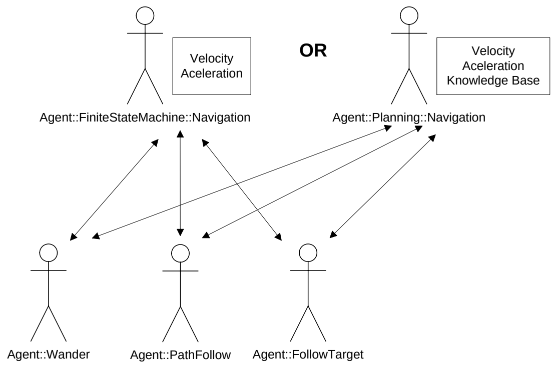


Fig. 8. Alternative navigation agents.

the agent representation for the navigation agent (FiniteStateMachine::Navigation). One possible alternative would be to handle navigation as a planning problem, e.g., logic-based. In that case, the FiniteStateMachine::Navigation agent would be replaced by a Planning::Navigation agent.

Specific rules must be followed while building the complete task coordination system. In the example given, all the navigation agents must share and update the continuous state of the robot in the blackboard. The position can be obtained by a **Sensor Fusion** agent, including several agents that run image, sonar, laser and processing algorithms, all living within the same robot or even in different team robots.

IV. Conclusions and Prospective Work

This paper described the main goals and reference scenario of the RESCUE - Cooperative Navigation for Rescue Robots project, currently being carried out by three research groups of ISR/IST. The project aims at introducing research advances in its enabling disciplines, namely as Computer Vision, Robot Navigation, Hybrid/Discrete Event Systems and Artificial Intelligence, applied to outdoors scenarios, mainly search and rescue operations.

Currently, two robots are being used by the project team. Preliminary experimental results obtained with the land robot using a reduced number of features for topological localization and mapping were presented for indoor and structured environments. According to the robot capabilities, the future work includes: i) the test of the algorithm using a larger number of features extracted from the land vehicle and aerial vehicle sensors cooperative navigation - (e.g., free space, environment sharpness, geometric features, speed, orientation), ii) the extensive testing in outdoor environments, iii) the integration of the Markov Model approach for localization presented in [17] in a simultaneous localization and mapping procedure in unstructured environments aiming at search and rescue operations.

A motion control library has been developed for the blimp aerial robot which provides the basic functions for its navigation. Line following, target tracking and formation control algorithms will be subjects for future work.

Underlying the whole project development is a software architecture that supports a multi-agent multi-robot functional architecture with several modes of operation. The architecture is based on a hierarchy of agents distributed across and within the team robots. The software architecture design is finished and the implementation work is currently on going.

Acknowledgments

The authors wish to express their acknowledgment to the student members of the RESCUE project team who contributed to the concepts, diagrams and results described in this project overview paper, namely Alexandre Bernardino (optical flow), João Frazão (software architecture), Thomas Krause (blimp motion control library) and Alberto Vale (land robot topological localization and mapping).

References

[1] H. Bruyninckx. "OROCOS: Design and Implementation of a Robot Control Software Framework", Proc. of IEEE ICRA 2002, April, 2002.

[2] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. "Design Patterns: Elements of Reusable Object Oriented Software", Addison-Wesley, Reading, MA, 1995.

[3] N. Gracias, J. Santos-Victor. "Underwater Mosaicing and Trajectory Reconstruction using Global Alignment", IEEE OCEANS 2001, Honolulu, November 2001.

[4] Y. Hur, I. Lee. "Distributed Simulation of Multi-Agent Hybrid Systems", IEEE International Symposium on Object-Oriented Real-time Distributed Computing (ISORC), April 29-May 1, 2002.

[5] M. Kijima. "Markov Processes for Stochastic Modeling", Chapman & Hall, (1997).

[6] H. Kitano, S. Tadokoro, I. Noda, H. Matsubara, T. Takahashi, A. Shinjou, S. Shimada. "RoboCup Rescue: Search and Rescue in Large-Scale Disasters as a Domain for Autonomous Agents Research", Proc. of IEEE International Conference on Man, Systems and Cybernetics, 1999.

[7] K. Konolidge. "Saphira Robot Control Architecture Version 8.1.0", SRI International, April, 2002.

[8] P. Lima, R. Ventura, P. Aparício, L. Custódio. "A Functional Architecture for a Team of Fully Autonomous Cooperative Robots", RoboCup-99: Robot Soccer World Cup III, Lecture Notes in Computer Science. Springer-Verlag, Berlin, 2000.

[9] D.C. MacKenzie and R.C. Arkin. "Evaluating the Usability of Robot Programming Toolsets", The International Journal of Robotics Research, Vol. 17, No. 4, pp 381-401, 1998.

[10] A. Papoulis. *Probability, Random Variables and Stochastic Processes*, McGraw-Hill, 1991.

[11] L. Rabiner. "A Tutorial on Hidden Markov Models and Selected Applications", Proc. of the IEEE, pp. 257–286, 1989.

[12] P. Tabuada, G. J. Pappas P. Lima. "Feasible Formations of Multi-Agent Systems", Proc. of the American Control Conference, Arlington, VA, 2001.

[13] S. Thrun. "Probabilistic Algorithm in Robotics", Artificial Intelligence Mag., **21(4)**, pp. 93–109, 2000.

[14] S. Thrun, W. Burgard and D. Fox, "A Real-Time Algorithm for Mobile Robot Mapping with Applications to Multi-Robot and 3D Mapping", Proc. of the IEEE Int. Conf. on Robotics and Automation, 2000.

[15] S. Thrun, W. Burgard and D. Fox, "A Probabilistic Approach to Concurrent Mapping and Localization for Mobile Robots", Machine Learning **31**, pp. 29–53, (1998).

[16] A. Vale, M. I. Ribeiro. "Environment Mapping as a Topological Representation", 11th International Conference on Advanced Robotics *(submitted)*, 2003.

[17] A. Vale, M. I. Ribeiro. "A Probabilistic Approach for the Localization of Mobile Robots in Topological Maps", Proc. of the 10th IEEE Mediterranean Conf. on Control and Automation, 2002.

[18] S. van der Zwaan, A. Bernardino, J. Santos-Victor. "Vision based station keeping and docking for floating vehicles", Proc. of the European Control Conference 2001, Porto, Portugal, September 2001.