A component-based approach to the design of networked control systems

Karl-Erik Årzén, Antonio Bicchi, Gianluca Dini, Stephen Hailes, Karl H. Johansson, John Lygeros, and Anthony Tzes

Abstract— **Component-based techniques revolve around composable, reusable software objects that shield the application level software from the details of the hardware and low-level software implementation and vice versa. Components provide many benefits that have led to their wide adoption in software and middleware developed for embedded systems: They are well-defined entities that can be replaced without affecting the rest of the systems, they can be developed and tested separately and integrated later, and they are reusable. Clearly such features are important for the design of large-scale complex systems more generally, beyond software architectures. We propose the use of a component approach to address embedded control problems. We outline a general component-based framework to embedded control and show how it can be instantiated in specific problems that arise in the control over/of sensor networks. Building on the middleware component framework developed under the European project RUNES, we develop a number of control-oriented components necessary for the implementation of control applications and design their integration. The paper provides the overview of the approach, discusses a real life application where the approach has been tested and outlines a number of specific control problems that arise in this application.**

I. INTRODUCTION

Networked embedded systems play an increasingly important role and affect many aspects of our lives. By enabling embedded systems to communicate, new applications are being developed in areas such as health-care, industrial automation, power distribution, rescue operations and smart buildings. Many of these applications will result in a more efficient, accurate and cost effective solution than previous ones. The European Integrated Project Reconfigurable Ubiquitous Networked Embedded Systems (RUNES) [8] brings together 21 industrial and academic teams in an attempt to enable the creation of large scale, widely distributed, heterogeneous networked embedded systems that inter-operate and adapt to their environments. The inherent complexity of such systems must be simplified if the full potential for networked embedded systems is to be realized. The RUNES project aims to develop technologies (system architecture, middleware, networking, control etc.) to assist in this direction, primarily from a software and communications standpoint.

Networked control systems impose additional requirements that arise from the need to manipulate the environment in which the networked systems are embedded. Timing and predictability constraints inherent in control applications are difficult to meet in general, due to the variations and uncertainties introduced by the communication system: delays, jitter, data rate limitations, packet losses etc. For example, if a control loop is closed over a wireless link, it should tolerate lost packets and be able to run in open loop over periods of time. Resource limitations of wireless networks also have important implications for the control design process, since limitations such as energy constraints for network nodes need to be integrated into the design specifications. The added complexity and need for re-usability in the design of control over wireless networks suggests a modular design framework.

In this paper, we propose a component-based approach to handle the software complexity of networked control systems. A general framework is presented and it is shown how it can be instantiated in specific problems that arise in control over wireless sensor networks as well as in control of network and communication resources. The proposed component framework hides network programming details from the control system designer. The components are well-defined entities that can be replaced without affecting the rest of the systems. It is shown how they can be developed and tested separately and integrated later. Building on the middleware component framework of RUNES, we develop a number of control-oriented components necessary for the implementation of control applications and design their integration. The paper provides the overview of the approach, discusses a real life application where the approach can be used and outlines a number of specific control problems that arise in this application. Companion papers [27], [12], [14], [15] provide the details of the implementation of specific components to address these control problems, as well as experimental validation results.

The paper starts by presenting a brief overview of middleware and component frameworks in general, and those targeted to networked embedded systems in particular (Section II). The RUNES tunnel disaster relief scenario that

Research supported by the European Commission under the integrated project RUNES, FP6-IST-004536.

K.-E. Årzén is with the Department of Automatic Control, Lund University, Sweden.

A. Bicchi is with the Department of Electrical Systems and Automation and Centro I.R. *E. Piaggio*, University of Pisa, Italy.

G. Dini is with the Department of Information Engineering, University of Pisa, Italy.

S. Hailes is with the Department of Computer Science, University College, London, U.K.

K. H. Johansson is with the School of Electrical Engineering, Royal Institute of Technology, SE-100 44 Stockholm, Sweden; kallej@ee.kth.se. Corresponding author.

J. Lygeros is with the Automatic Control Laboratory, ETH Zurich, Switzerland.

A. Tzes is with the Department of Electrical and Computer Engineering, University of Patras, Greece.

serves to focus our work is then described in Section III. An overview of control problems that arise in the scenario is also given in the same section: maintaining the connectivity of a wireless sensor network in an adverse environment, utilizing the resources of the network itself (e.g., wireless transmission power control) and those of mobile robots (e.g., to replace missing nodes). In Section IV we discuss the components that need to be implemented to address the specific control problems in the task on physical network reconfiguration; the details of the development of these components and their experimental testing are given in the companion papers. Some examples of general purpose, low-level control components are also presented in the section. Section V details a security component framework, which provides an interface to protect communications among nodes. The hardware and software integration for the demonstration of the physical network reconfiguration is given in Section VI. Its validation is shown in Section VII, which describes both a computer simulation of the scenario and some preliminary experimental results. Some concluding remarks are given in Section VIII.

II. Middleware and Components

A. Middleware

In a component-based software system, a component is a system element offering a predefined service and able to communicate with other components. A component is a unit of independent deployment and versioning. It is encapsulated and non-context specific. It follows that components can interact with other components without knowing much of their internal structure or their execution environment (for example, their operating system or network protocols). Clearly, devising such an abstract level of interaction is a non-trivial effort. In many cases, an effective solution can be found by the judicious application of a software abstraction layer, known as middleware. Middleware mediates the interactions of a component with its environment by providing a programming interface transparent to the operating systems and to the network protocols underneath. A comprehensive survey of middleware concepts (motivated primarily for networked embedded systems) can be found in [9]. Important examples of middleware currently in use are Java Remote Method Invocation (Java RMI) [5], Microsoft Component Object Model (COM) [6], and Common Object Request Broker Architecture (CORBA) [2]. These frameworks, however, are not specifically targeted to embedded systems or distributed control systems. The resource constrained implementation platforms common in embedded and distributed control systems imply additional, severe requirements on the middleware. To meet these requirements, extensions of general purpose middleware have been developed. One such example is real-time CORBA [30], which features prioritized scheduling policies for threads and export some control parameters in the communication protocols. Even real-time CORBA, however, has several shortcomings that make its use on demanding embedded system applications problematic [9].

Several application domains have emphasized the importance of developing software infrastructures specifically tailored to the needs of the domain. For example, the automotive industry has formed the development partnership AUTOSAR [1], to achieve modularity, scalability, transferability and re-usability of software functions in vehicles. AUTOSAR strives to provide an open system architecture for automotive systems based on standardized interfaces for the different system layers. A precise component definition and an appropriate composition framework are essential to answer a variety of questions on system architectures, e.g., on synchronization and network protocols [25].

Specific control and real-time requirements on the middleware have also been investigated in recent academic software prototypes. Etherware [16] is a middleware for networked control that was recently proposed. This middleware focuses on the ability to maintain communication channels during component restarts and upgrades and to recovery from failure situations. ControlWare [31] is a middleware that utilizes feedback control for guaranteeing performance in software systems. Though not specifically targeted to embedded systems, its usefulness has been demonstrated on web server and proxy quality of service management. A tutorial overview of software technologies for reusable and distributed control systems is given in [23].

Finally, from a theoretical point of view, semantic frameworks that support composition and abstraction operations are central to the formal modeling and analysis of such distributed systems. For embedded systems (where the logic functions encoded in the computational elements have to interact with a primarily analog environment) the most relevant frameworks are those developed in the area of hybrid systems. Several such frameworks have been proposed in recent years, to support the modeling, verification, system development and simulation efforts; for an overview see [29]. Some are general purpose, while others are targeted to specific application areas [18]. Most are also supported by simulation, verification or design computer tools. A link between these theoretical developments and the middleware frameworks discussed above is just emerging as an exciting and important research area.

B. Component frameworks for networked embedded systems

The main reason for using component-based approaches in software development is to enforce re-usability. A new software application is built from existing well-tested components. The components are composed (or assembled) into applications. It is often possible to aggregate components together, forming new components.

Component-based software engineering has been successfully used in several software development projects, primarily for desktop and eBusiness applications. Within real-time embedded systems, the use of component techniques is not well-developed. For desktop applications the COM technology is most widely used. COM components are often relatively large in size, each component encompassing a substantial amount of the application functionality. Another widely used class of component models are the models that have their basis in distributed object models. These include the CORBA Component Model (CCM) [3], Enterprise Java Beans (EJB) [4], and .NET [7]. The .NET model can be viewed as an distributed evolution from COM that is especially interesting due to Common Language Runtime (CLR). CLR is a virtual machine technology that can be compared to Java's Virtual Machine. It is Microsoft's implementation of the Common Language Infrastructure (CLI) standard, which defines an execution environment for program code. The CLR executes a bytecode format into which several languages can be compiled, e.g., C# and Visual C++. Through this it is possible to integrate software components developed in different programming languages. The drawback with the approach, compared to, e.g., Java-based approaches, is that it is operating system dependent, i.e., it is only supported for Windows-based systems. Components are viewed as extended objects that can be distributed. However, each individual object still resides on a single node in the network. In these types of component models object-oriented concepts, such as classes and inheritance, are integral parts.

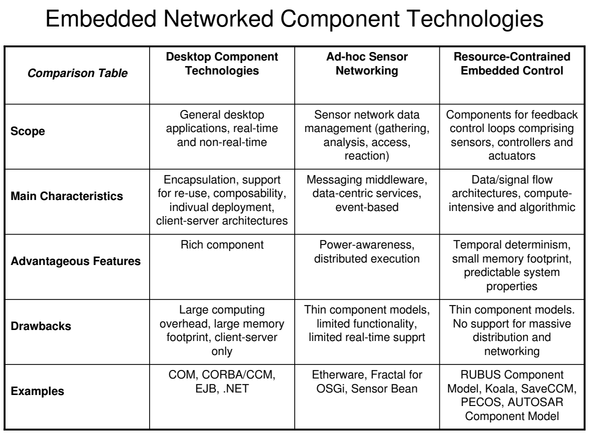
In component technologies for embedded systems, non-functional properties such as safety, timeliness, memory footprint, and dependability are of particular interest. Compared to the desktop component approaches described above the component models here are much more limited in functionality. Often the component models are intended for applications of an algorithmic nature. These applications are commonly modeled as data- or signal-driven block diagrams. Another name for this is a pipe and filter architecture. The individual components are typically smaller than in the previous component models, and the emphasis on component aggregation is larger. These component technologies are frequently inspired by the block diagram approach in Matlab/Simulink, the function block diagrams in the automation language standard IEC 61131-3, and by ordinary discrete logic gates. There are still no good examples of commercially successful component technologies for embedded systems. However, it is an area where considerable research currently is being performed.

For sensor network and mobile ad-hoc network applications, all the component technologies above are, in principle, applicable. Sensor networks are an example of a severely resource-constrained distributed implementation platform. If they are to host sensor fusion and control applications, it is quite clear that the component technologies developed for embedded systems are a natural option. Embedded control systems and sensor network applications, furthermore, have many similarities. In both cases, a component model centered around data flows is more natural than the focus on component function calls found in desktop component models. Following this path a possibility would be to develop a set of generic sensor, data fusion, control and actuator components or component types; examples along these lines are outlined in Section IV. The limited battery resources make power-awareness an important attribute of component models for sensor networks.

The different characteristics of desktop applications and sensor/actuator networks do, however, not necessarily imply that it is not possible to base a component model for sensor/actuator networks on more conventional component technologies; the development effort only becomes considerably larger. Rather than having built in support for data flows in the middleware, it has to be explicitly realized through component function calls. This is the approach that has been taken in the RUNES project.

In mobile ad-hoc network applications the resource constraints are normally less severe than in wireless sensor networks. More powerful CPUs with more memory and battery resources are often used. Hence, here the desktop-type of component technologies can be applied. The components of this type are often more application-oriented than the simpler and more generic sensor-controller-actuator components. In a mobile robot setting we may decompose the application into components for localization, path planning, collision avoidance, etc. These are the types of components of main interest the work presented here.

The table in Figure 1 summarizes embedded networked component technologies by listing their characteristics together with some advantages and disadvantages.


Embedded Networked Component Technologies

Comparison Table	**Desktop Component Technologies**	**Ad-hoc Sensor Networking**	**Resource-Contrained Embedded Control**
Scope	General desktop applications, real-time and non-real-time	Sensor network data management (gathering, analysis, access, reaction)	Components for feedback control loops comprising sensors, controllers and actuators
Main Characteristics	Encapsulation, support for re-use, composability, indivual deployment, client-server architectures	Messaging middleware, data-centric services, event-based	Data/signal flow architectures, compute-intensive and algorithmic
Advantageous Features	Rich component	Power-awareness, distributed execution	Temporal determinism, small memory footprint, predictable system properties
Drawbacks	Large computing overhead, large memory footprint, client-server only	Thin component models, limited functionality, limited real-time supprt	Thin component models. No support for massive distribution and networking
Examples	COM, CORBA/CCM, EJB, .NET	Etherware, Fractal for OSGi, Sensor Bean	RUBUS Component Model, Koala, SaveCCM, PECOS, AUTOSAR Component Model

Fig. 1. Comparison of embedded networked component technologies.

C. *RUNES middleware components*

Central to our efforts in developing a component-based framework for networked control is the RUNES middleware component model [10]. Even though sensor networks (and other ad hoc networks) are of central interest to RUNES, the RUNES middleware component model is closer in spirit to the desktop model discussed above, than to the embedded model. One reason for this is that the RUNES components are not only intended for the sensor nodes, but should also reside on the gateways and on the back-end computers. Another reason is that the RUNES components are also intended as a means for structuring parts of the RUNES middleware itself.

A component-based framework for networked control should enable quality of service definitions and negotiation between the designer of the control application and the middleware. The solution should combine the appropriate level of abstraction needed by control applications with a lightweight and scalable architecture. The middleware should provide the appropriate support for a wide variety of control applications, ranging from sensor networks to distributed control systems. To this end, it is of utmost importance to keep track of the level of introduced complexity. Memory consumption and communication latency are examples of fundamental parameters in the design. Our conclusion is that, even if some existing proposals attempt to cope with some of these issues, a middleware based on a comprehensive evaluation of the multifaceted requirements of networked control applications is still to come.

The RUNES middleware [10] is illustrated in Figure 2. The middleware acts as a glue between the sensor, actuator, gateway and routing devices, operating systems, network stacks, and applications. It defines standards for implementing software interfaces and functionalities that allow the development of well-defined and reusable software. The basic building block of the middleware developed in RUNES is a software component. From an abstract point of view, a component is an autonomous software module with well-defined functionalities that can interact with other components only through interfaces and receptacles. Interfaces are sets of functions, variables and associated data types that are accessible by other components. Receptacles are required interfaces by a component and make explicit the inter-component dependencies. The connection of two components occurs between a single interface and a single receptacle. Such association is called binding and is shown in more detail in Figure 6. Part of the RUNES middleware has been demonstrated to work well together with the operating system Contiki [22], which was developed for low-memory low-computation devices. The implementation of the component model for Contiki is known as the component runtime kernel (CRTK). This component framework provides for instance dynamic run-time bindings of components, i.e., during execution it allows components to be substituted with other components with the same interface.

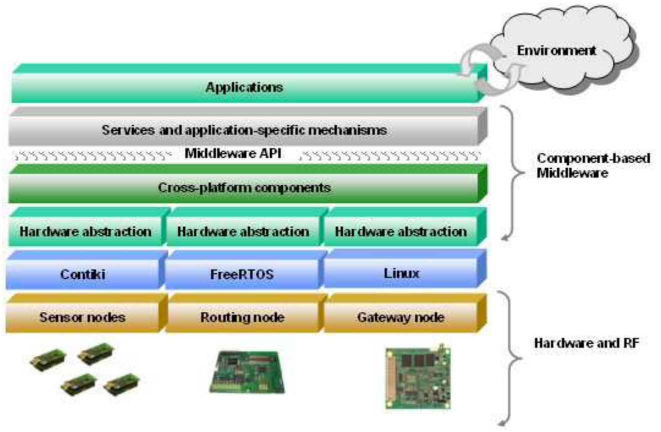


Fig. 2. Overview of the RUNES middleware platform. The component-based middleware resides between the application and the operating systems of the individual network nodes.

III. Motivating scenario

This section describes the RUNES tunnel disaster relief scenario and gives an overview of some of the control problems that arise within the scenario.

A. Disaster relief scenario

One of the major aims of the RUNES project is to create a component-based middleware that is capable of reducing the complexity of application construction for networked embedded systems of all types. Versions of the component runtime kernel, which forms the basis of the middleware, are available for a range of different hardware platforms. However, the task is a complex one, since the plausible set of sensing modalities, environmental conditions, and interaction patterns is very rich. To illustrate one potential application in greater detail, the project selected a disaster relief scenario, in which a fire occurs within a tunnel, much as happened in the Mont Blanc tunnel in 1999. In this, the rescue services require information about the developing scenario both before arrival and during rescue operations, and such information is provided by a network of sensors, placed within the tunnel, on robots, and on rescue personnel themselves. We explore the scenario in more detail below, but it should be noted this is intended to be representative of a class of applications in which system robustness is important and the provision of timely information is crucial. So, for example, much the same considerations apply in the prevention of, or response to, Chemical, Biological, Radiological, Nuclear or Explosive (CBRNE) attacks; likewise, search and rescue operations, and even industrial automation systems form application domains with similar requirements for predictability of response given challenging external conditions.

The fire-in-a-tunnel scenario deals with disaster relief activities in response to a fire in a road tunnel caused by an accident, as illustrated in Figure 3. For example, in the case of Mont Blanc, a very severe fire was caused as the result of the ignition of a lorry carrying margarine and flour. The resulting fire burned for two days, trapping around 40 vehicles in dense, poisonous, smoke, with a death toll of 37 people. Communications, lighting, and sprinkler systems failed within minutes of the fire starting with the result that Christian Comte, fire brigade chief at Chamonix, is reported to have said: *Sur le moment, on n'avait pas d'informations précises—on ne savait pas ce qui brûlait, ni à quel endroit, s'il y avait du monde à l'intérieur ou pas.* In other words, there was no precise information about what was happening: it was not clear what was burning, nor where it was, nor whether there were people inside the tunnel or not. As a consequence, firefighters entered the tunnel long past the time at which they could have made a difference, and themselves became trapped.

In the RUNES scenario, we project what might happen in a similar situation if the vision of the US Department of Homeland Security's SAFECOM programme becomes

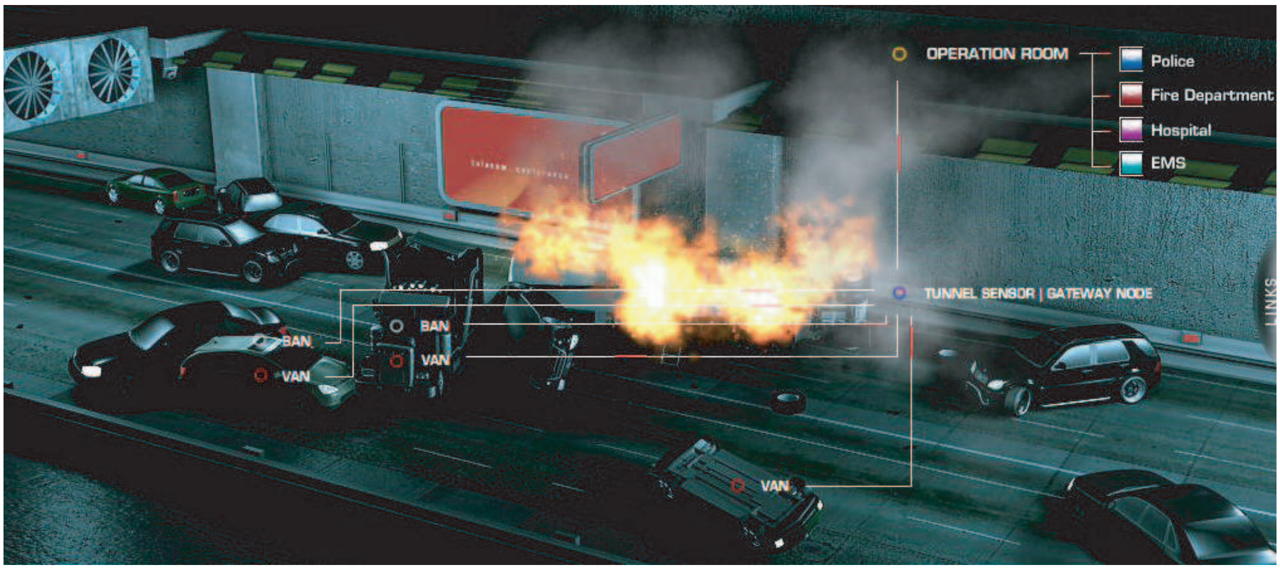


Fig. 3. Illustration of the RUNES tunnel disaster relief scenario.

a reality. The scenario is based around a storyline that sets out a sequence of events and the desired response of the system, part of which is as follows. Initially, traffic flows normally through the road tunnel; then an accident results in a fire. This is detected by a wired system, which is part of the tunnel infrastructure, and is reported back to the Tunnel Control Room. The emergency services are summoned by Tunnel Control Room personnel. As a result of the fire, the wired infrastructure is damaged and the link is lost between fire detection nodes (much as happened in Mont Blanc). However, using wireless communication as a backup, information from (for example) fire and smoke sensors continues to be delivered to the Tunnel Control Room seamlessly. The first response team arrives from the fire brigade and rapidly deploys search and rescue robots, following on foot behind. Each robot and firefighter carries a wireless communication gateway node, sensors for environmental temperature, chemical and smoke monitoring, and the robots carry light detectors that help them identify the seat of the blaze.

The role of the robots in this scenario is twofold: to help identify hazards and people that need attention, without exposing the firefighters to danger; and to augment the communications infrastructure to ensure that both tunnel sensor nodes and those on firefighters remain in contact with the command and control systems that the situation commander uses to make informed decisions about how best to respond. To accomplish this, the robots are moving autonomously in the tunnel taking into account information from tunnel sensors about the state of the environment, from a human controller about overall mission objectives, and from received signal strength measurements from the wireless systems of various nodes about the communication quality. The robots coordinate their activity with each other through communication over wireless links. Local backup controllers allow the robots to behave reasonably in the event that communication is lost.

B. *Overview of control problems*

The RUNES work in general and the disaster relief scenario in particular offer a number of interesting and challenging problems where control methods can make a key contribution. One can envision control algorithms being developed to control infrastructure resources; such as fans or fire extinguishing devices, control robot motion in order to localize hazards or localize injured humans and assist in removing them from the disaster area, and, last but not least, control network resources to ensure connectivity and timely delivery of crucial information. Here we will focus our attention to this last type of control problem, namely controlling network resources.

The control problem of interest is sketched in Figure 4. A set of nodes with wireless communication capabilities are deployed inside the tunnel for monitoring purposes. As soon as an emergency situation occurs, for example an accident involving many cars, the nodes need to transmit data regarding the tunnel conditions to a base station. In such a scenario, accurate and comprehensive information must be provided to the base station so that correct counter measures can be taken. It is of fundamental importance that the network would maintain connectivity, so that the flow of critical data to the base station is guaranteed. However, the network could be partitioned because of a malfunction of the nodes, caused by a fire, or because the presence of obstacles that deteriorate or even nullifies metrics of the Quality of Service.

In such a critical situation, the control application is responsible for restoring the network connectivity. This is done by sending a mobile autonomous robot inside the tunnel, see Figure 4. The robot is equipped with a radio transmitter–receiver so that it can maintain connectivity with

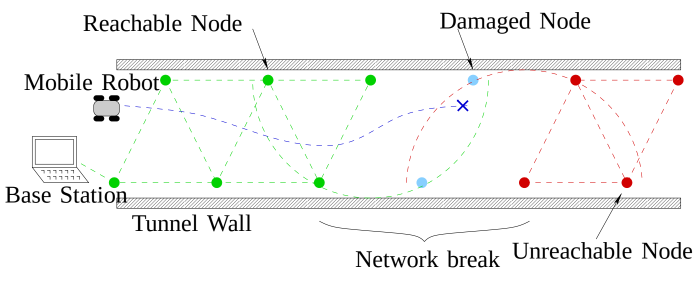


Fig. 4. Road tunnel scenario in which part of the deployed wireless network is disconnected due to two damaged network nodes. A mobile robot moves into the region of the damaged nodes to relay information from the unreachable nodes towards the base station.

the base station directly or through the deployed network. Once the base station determines the network break area, a target position for the mobile robot is computed. This is done by the network reconfiguration component. The robot then needs to navigate inside the tunnel until either it reaches the target position or it determines that the target position is out of reach because of obstacles.

Control applications impose additional requirements on the RUNES platform that arise from the need to manipulate the networked systems and/or the environment in which they are embedded. In the rest of the paper we present the organization of the control system components that need to be implemented in order to guarantee that network connectivity is reestablished. The core are the four components: network reconfiguration, localization, collision avoidance and power control. Details of the development of these components are given in the companion papers.

IV. Control Components for Maintaining Network Connectivity in Adverse Environments

This section describes the software architecture for the control components used for maintaining network connectivity, together with the functionality of each component. The control components outlined below follow the RUNES component model [10]. The four main control components deal with network reconfiguration, localization, collision avoidance and power control. Their integration is demonstrated through the network reconfiguration scenario described next. The section concludes with a discussion of the low-level component library containing sensor, data fusion, controller and actuator components; the higher level components of network reconfiguration, localization, collision avoidance and power control invoke the low level components in this library to accomplice their goals. Communication security issues are addressed by a specialized security component (which in turn comprises a number of subcomponents); this component is dealt with separately in Section V.

A. *Physical network reconfiguration scenario*

Mobile autonomous robots are sent inside the tunnel to restore connectivity, see Figure 4. The navigation of a robot inside the tunnel is made possible by two components. The first is the localization component, that provides the position and orientation of the robot inside the tunnel and information

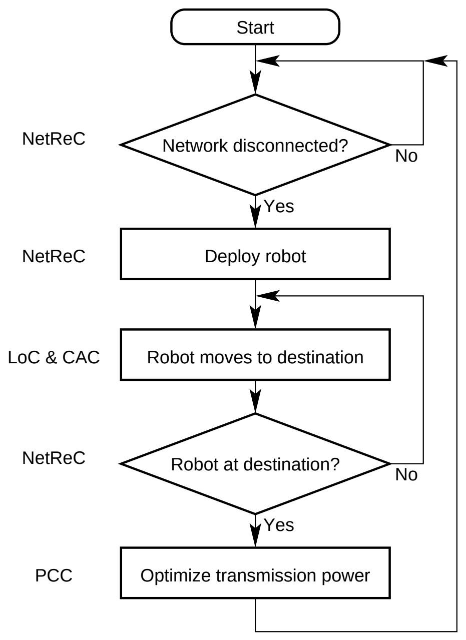


Fig. 5. Flow chart showing the actions taken in order to reestablish network connectivity. The acronyms to the left indicate the active control components.

about the presence of obstacles. The second is the collision avoidance component that ensures that the robot does not collide with obstacles or other robots. Once the mobile robot is in a suitable position it attempts to reconnect the network, by acting as a relaying node between the nodes in the disconnected parts of the network. At this stage, a third component, the power control component, is invoked, to reduce the energy consumption and lower the packet collision probability of the nodes at the boundary of the network. In case the network is not reconnected with the first robot, additional robots could be deployed in a similar fashion.

The flowchart in Figure 5 details the sequence of tasks in the reconfiguration scenario. The acronyms in the column to the left indicate the control component primarily responsible for the action. The scenario starts by the detection of that the network is disconnected. The network reconfiguration component (NetReC) then makes the decision that the first robot should be deployed. The robot moves autonomously to the destination using localization information about its position provided by the localization component (LoC). In parallel, it also uses the collision avoidance component (CAC) to avoid colliding with stationary objects or other moving agents. When the network reconfiguration component detects that the robot has reached a suitable goal position (possibly by adjusting the original destination point based on local information at the scene), it initializes the power control component (PCC). The radio transmission power is adjusted in the robot node and in its neighboring network nodes, in order to not only preserve battery power but also minimize interference among nodes. If the network is still disconnected after the power has been adjusted, the algorithm starts over and a new robot is deployed.

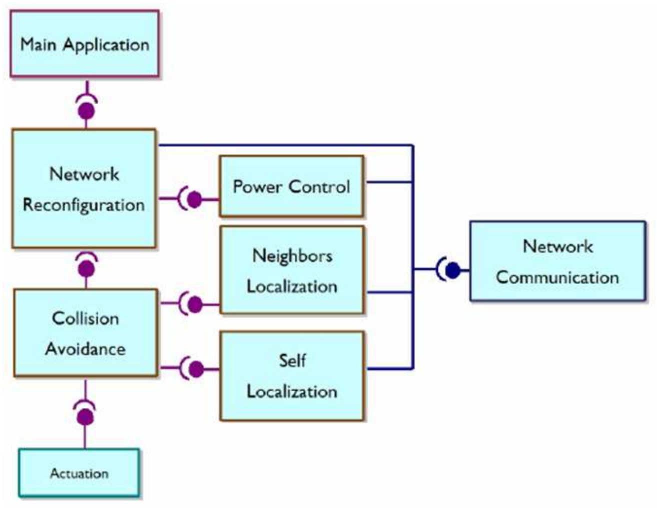


Fig. 6. Component framework layout for the control components used for maintaining network connectivity.

The interrelations between the control components and the overall application, robot actuation platform, and communication network are indicated in the component framework layout shown in Figure 6. The components are encapsulated software units of functionality and deployment that interact with other components exclusively through interfaces and receptacles [10]. The figure shows that the main application, which initializes the restoration of the network connectivity, interacts with the network reconfiguration component. This component supervises activity through its coupling to the collision avoidance and the power control components. The collision avoidance component is responsible for the physical actuation of the commands, i.e., for moving the robot. It bases its decisions on information from the localization component, which is divided into one part handling the localization of neighboring robots and other potential obstacles, and another part providing localization information about the robot itself. As indicated in the figure, the control components rely heavily on the network communication.

B. *Network reconfiguration component*

The network reconfiguration component is responsible to position the mobile robot at a point that ensures the restoration of the communication along the network. The network reconfiguration component is activated as soon as information about that the network is disconnected and the localization of the malfunctioning network nodes has been obtained. The functionality of the component is based on a beacon, which tests the operational capabilities of each node near the network break area through handshaking. The connectivity of the mobile robot with the operational part of the network is rather critical, because commands may be sent from the base station to the robot and information from the robot can be requested. The locations of the (static) network nodes are assumed to be known a priori.

The network reconfiguration component provides the goal coordinates to the collision avoidance component, which ensures that the robot safely moves to the vicinity of the furthest known operational node of the network. The component computes the area, within which the final positioning of the robot is possible. This area is important in the case obstacles have occupied the damaged node's position or do not allow for a closer visit. From the obtained position, the robot tries to establish communication with both the network connected to the base station and the network that has been disconnected. The former action is performed in order to check the functionality of the specific node, but also to ensure that the communication failure has not been due to debris or other kinds of communication blocks. If the functionality of the node is verified, the robot moves on to the next node, etc., until the non-operational node is accurately located. As soon as this node is located, the network reconfiguration component provides the necessary inputs for the initialization of the power control component. If the non-operational node cannot be located through an exhaustive search, there is a change in mode of operation into finding a sub-optimal position within the lattice of the sensor nodes. If even this approach fails, the mobile robot is positioned near the edge of the previously described region and sends a message to the base station for the deployment of a second mobile robot. The algorithm terminates as soon as the communication coverage of the region of interest is completed. The network reconfiguration component is further described in [27].

C. *Localization component*

The localization component provides interfaces to localize mobile agents or robots. It also provides interfaces to detect obstacles in front of the mobile robot. Each mobile robot is required to contain one instance of this component. Additionally, each of the stationary sensor motes within the tunnel are required to contain one instance of the complementary distance sensor component.

The localization method is based on ultrasound. Each mobile robot is equipped with an ultrasound sender unit and each stationary sensor mote is equipped with an ultrasound receiver unit. The mobile robot periodically broadcasts a radio message shortly followed by an ultrasound pulse. Each stationary node measures the difference in time of arrival between the radio message and the ultrasound pulse, and uses this to calculate its distance to the mobile robot. After a predetermined time to avoid contention, a radio packet containing the distance is sent to the mobile robot. After emitting the ultrasound pulse, the mobile robot spends a predetermined time collecting distance measurements from the sensor nodes. After that, data fusion is applied to the collected distance measurements. Different alternatives have been evaluated. One possibility is to use triangulation. In this approach as many triangular sensor cells as possible are formed, and a position estimate is calculated for each cell. Finally, the individual position estimates are combined into a single estimate using outliers removal and averaging. Once a position estimate is available this is used as an input to the corrector part of an extended Kalman filter. The predictor part of the filter uses the encoder information from the robot wheels to predict its current position. Alternatively, it is possible to directly use the estimated distances as inputs to the extended Kalman filter. Both approaches require that each mobile robot knows the position of every stationary sensor node.

The above approach runs the risk of not working when multiple robots simultaneously try to determine their locations. In order to avoid this problem the mobile robots also listen to the radio messages associated with ultrasound messages. If a mobile robot hears this type of message it will wait, or back off, a certain time, before attempting again to emit its combined radio and ultrasound pulse.

Once the self-localization is performed, i.e., the filtered position and orientation estimates are available, a radio packet containing this information together with a time-stamp is multi-casted to the other mobile robots. Thus each mobile robot obtains information about the current location of its neighboring mobile robots.

The ultrasound localization can only be used to detect the position of known mobile robots. In order to detect unknown obstacles, the mobile robots need to be equipped with one or several forward pointing proximity or distance sensors. One possibility is to use an IR sensor, e.g., a SHARP GP2D12 sensor. Either one or two fixed sensors are used, or a sensor is mounted on a simple RC servo, which then sweeps a certain angular region in front of the mobile robot. The localization component is further described in [12].

D. Collision avoidance component

The collision avoidance component provides an interface to steer, in a safe way, a mobile robot to a desired final position with an assigned heading. The collision avoidance strategy is based on a *reserved disk* associated to each robot. The disk contains all the positions that can be reached if the vehicle performs a maximum curvature turn in clockwise direction.

The motion strategy of the robot is based on four distinct modes of operation, each assigning a suitable value to the curvature rate of the robot. Figure 7 shows these modes along with the corresponding switching conditions. The robot enters the straight mode if is possible to move in the same direction as the robot is heading, i.e., if its reserved disk does not overlap with other reserved disks. When the robot is in this mode, its curvature rate is set to zero. Whenever its reserved disk becomes tangent to the one of another robot, a test is made based on the current motion heading θ. If a further movement in the direction specified by θ causes an overlap, then the robot enters the hold mode. Otherwise, the robot is able to proceed, and remains in the straight mode. When the hold mode is entered, the robot's curvature rate is set to the minimum allowable, and the motion of its reserved disk is stopped. As soon as the robot motion in the heading direction is permitted but not directed towards the target destination, the robot enters the roll mode, and tries to go around the reserved disk of the other robot. This is achieved by selecting a suitable value for the curvature rate

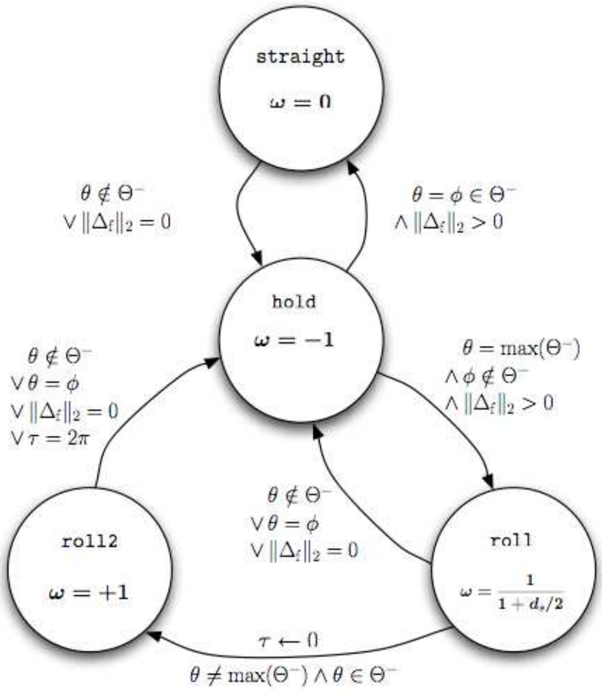

straight
$\omega = 0$
$\theta \notin \Theta^-$ $\vee \|\Delta_\xi\|_2 = 0$
$\theta = \phi \in \Theta^-$ $\wedge \|\Delta_\xi\|_2 > 0$
hold
$\omega = -1$
$\theta \notin \Theta^-$ $\vee \theta = \phi$ $\vee \|\Delta_\xi\|_2 = 0$ $\vee \tau = 2\pi$
$\theta = \max(\Theta^-)$ $\wedge \phi \notin \Theta^-$ $\wedge \|\Delta_\xi\|_2 > 0$
$\theta \notin \Theta^-$ $\vee \theta = \phi$ $\vee \|\Delta_\xi\|_2 = 0$
roll2
$\omega = +1$
roll
$\omega = \frac{1}{1 + d_s/2}$
$\tau \leftarrow 0$
$\theta \neq \max(\Theta^-) \wedge \theta \in \Theta^-$

Fig. 7. Finite state automaton that summarizes the collision avoidance protocol implemented by the collision avoidance component.

of the robot such that the two disks never overlap. During roll, the tangency of the two disks can unexpectedly be lost. In such circumstances, the robot enters the roll2 mode, and the curvature rate is set to the maximum allowable in order to restore the contact. The roll2 mode can only be entered if the previous mode was roll. When the tangency is restored, the robot switches back to the hold mode and possibly from there to the roll mode again.

The decentralized characteristic of the collision avoidance protocol allows the collision avoidance component to be implemented on-board the robots. Each robot is able to make a safe decision about its motion, based only on locally available information. This information consists of the position and orientation of robots that are within a certain sensing or communication radius. For this reason, each robot is not required to explicitly declare its positioning goal. The collision avoidance component is further described in [12].

E. Power control component

The power control component provides an interface for regulating the transmission power of nodes at the boundary of a disconnected area of the wireless network of the tunnel. The main functionality of this component is to provide a power control algorithm that adjust the power such that the network is reconnected. A fine tuning of the output power is essential to preserve the battery of the nodes and to minimize interference among network nodes.

The power control algorithms are based on radio models for the network nodes, i.e., the Telos motes. Communication quality is characterized through the received signal strength

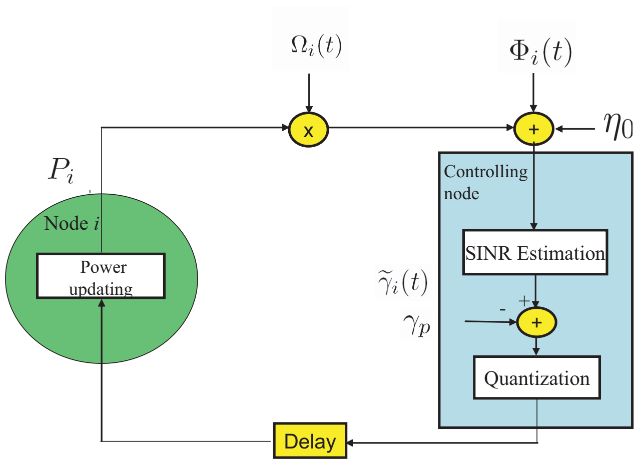


Fig. 8. The transmission powers of the network nodes are controlled by the power control component residing in the mobile robot. Each power control loop consists of a transmitter node (on the left) and a receiver node (on the right). The receiver node executes the control algorithm and sends the command to the transmitter.

indicator (RSSI), which is related to the signal to noise plus interference ratio (SINR) for the radio devices mounted on the nodes. The power control mechanism reacts to the fluctuations of the SINR by controlling the level of power that the transmitter uses in order to ensure a desired packet error probability. The power control command is computed in the receiver and then communicated to the transmitter, which transmits packets using the updated power level. The feedback control loop for a single transmitter–receiver pair is shown in Figure 8.

Two power control mechanisms have been developed and tested: one scheme based on a multiplicative-increase additive-decrease (MIAD) updated of the power, and one scheme that is based on a model of the average packet error rate (PER). The MIAD power control, which is inspired by [11], implements the following simple algorithm: each correctly received packet imposes a decrease of the transmit power by Δ, where Δ is the step size; whereas when an erroneous packet is detected, the transmission power is increased by $d\Delta$, where d is a positive integer. The parameters d and Δ influence performance (packet error rate and power consumption), and thus need to be tuned. The PER power control adjust the transmission power according to a model of what power is needed for a desired SINR given the current SINR and power. Accurate estimates of SINR and RSSI are obtained through online filters. The power control component is further described in [14].

F. Low-level control component library

The components defined above build on and make use of lower-level components developed to perform tasks such as polling sensors, sending commands to actuators, etc. We conclude this section by providing a brief overview of some examples of this type of components.

Sensor components: Two types of sensor components can be distinguished. A passive sensor component returns a physical measurement, e.g., temperature, light intensity, etc, through the use of some hardware device, as the response to a call to an interface function, e.g., `getValue()` from either another component on the same node, from the application code in the node, or from a component or application on some other node via the radio interface. Hence, from a passive sensor component, measurement values must be pulled by the users of the component

An active sensor, on the contrary, is realized by a separate execution thread that provides a new measurement value on its own initiative. This value is then forwarded to, e.g., another component by a call to the corresponding interface function of that component. Hence, the active sensor pushes new values to the users of the component.

Another distinguishing characteristic of sensor components is whether they are time-driven or event-driven. In a time-driven sensor the measurement is performed periodically. In an event-driven sensor the measurement is performed when an event occurs. This event could be the call to some `getValue` interface function. However, we could also think of an event-driven active sensor that, e.g., only generates a new output if its current value has changed sufficiently much from the old value. Also, it is not necessary for a passive sensor to be event-driven. A passive sensor could be realized as an execution thread that performs the sampling periodically, but where the users of the component still must pull out a value, in this case typically the latest value, by calling an interface function.

A sensor component could also contain other functionality. Instead of generating only a value it could associate the value with a time stamp indicating when the value was generated. The sensor may perform filtering, e.g., low-pass filtering. For an event-based sensor several different event types are plausible, together with the associated threshold values.

Data fusion components: In sensor network application data fusion or aggregation is important. A major reason for this is a desire to reduce the communication traffic in the network.

Because of the spatial distribution of the sensors, data fusion can be performed both in the time and space domain. An example of data fusion in the time-domain is down-sampling; for example, an active sensor may elect to only forward a reduced number of data values. This can be done periodically, e.g., removing every second data value, or be event-driven. In the latter case one could think of a data fusion component that only forwards data values that correspond to significant changes in the value. Other types of time-domain data fusion are various sorts of averaging and windowing. Data fusion in the space-domain shares several similarities with the time-domain. Spatial averaging is one example.

Another distinguishing characteristic of data fusion components is whether they are signal-based or model-based. A signal-based data-fusion component performs the aggregation using the signal values as the only information source. In model-based data fusion there is a model, e.g., a differential equation, that describes the spatial or temporal relationship between one or several measurements. Using this it is possible to refine the data aggregation. Techniques based on, e.g., Kalman filtering, can be used to estimate signals that are not measured.

A relevant technique for redundancy reduction in the information flow generated by the nodes of wireless sensor networks is distributed source data compression, e.g., [24], [19]. This technique compresses data based on the (usually significant) spatial and temporal correlation of the sensor measurements.

Controller components: A controller component realizes a dynamical system and comprises the actual control algorithm of the component. A distinguishing characteristic is whether the component operates on a single measured signal and generates a single controller output (SISO controller), or whether either or both the input and output consist of multiple signals (MIMO controller).

Another distinguishing feature is whether the controller is linear or non-linear. A general non-linear controller can be represented as

$$\begin{aligned} x_{k+1} &= f(x_k, y_k, r_k) \\ u_k &= g(x_k, y_k, r_k), \end{aligned}$$

where x is a vector representing the internal state of the controller, y is the measurement, r is the reference or setpoint for y, and u is the output of the controller. The subscript k indicates time step. If the functions f and g are linear, then the controller is linear and can be represented as

$$\begin{aligned} x_{k+1} &= Ax_k + By_k + Gr_k \\ u_k &= Cx_k + Dy_k + Er_k \end{aligned}$$

where $A, B, C, D, E,$ and G are matrices of suitable sizes. These general forms can be one possible starting point for a controller component library. Another possibility is to focus on commonly used special forms of the above. Some examples are PID controllers, state feedback controllers, observer-based controllers, and lead-lag controllers.

Actuator components: An actuator component determines the action the system takes on the physical environment. It shares some of the characteristics of the sensor component: it can be based on push, namely, the component provides at the interface a function, e.g., `setValue(val)`, which sets the value `val` to the actuator; or it can be based on pull, namely, the actuator itself requests the value. Furthermore, the actuation can be time-driven or event-driven. In the first case, the actuator component is accessed at precise time instances, whereas in the second case it is accessed when a predefined event is detected. The actuator component may include some filtering, such as a zero-order hold, or nonlinear filters, e.g., saturation.

V. Security

In a system like the one under consideration, protecting communication among mobile robots and sensor nodes poses unique challenges. First of all, unlike traditional wired networks, in a wireless network, an adversary with a simple radio receiver/transmitter can easily eavesdrop as well as inject or modify packets. Second, in order to make the system economically viable, nodes are limited in their energy, computation, storage, and communication capabilities. Furthermore, they typically lack adequate support for making them tamper-resistant. Therefore, the fact that nodes can be deployed over a large, unattended, possibly hostile area exposes each individual node to the risk of being compromised.

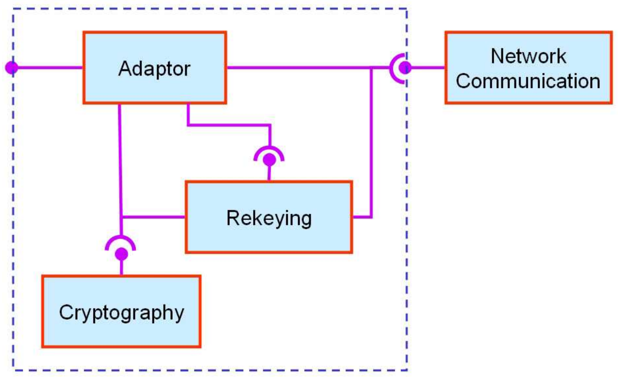


Fig. 9. The Security Component Framework.

In this wireless network, security hinges on a group communication model. This means that authorized nodes in the network share a symmetric group key that is used to encrypt communication messages. Anyone that is not part of the group can neither access nor inject or modify messages. This implies that when a node leaves the group, the current group key must be revoked and a new one distributed to all nodes except the one that is leaving (forward security). A node may leave the group either because it has finished its task or because it is considered malicious and thus it must be evicted. Failure to provide the correct group key can be interpreted as an alarm by the system, which triggers countermeasures. It follows that the ability to revoke keys translates into the ability to logically remove compromised nodes from the network.

The Security Component Framework provides an interface to protect communications among nodes and guarantee forward security. Figure 9 shows the architecture of the Security Component Framework in terms of components and their interrelations. The *Cryptographic component* provides an interface for the basic cryptographic primitives such as symmetric ciphers (e.g., Skipjack and RC5) and hash functions (e.g., SHA-1). The *Rekeying component* provides and interface for key distribution and revocation. This component implements cryptographic network protocols and, therefore, uses the services offered by the Network Communication component and the Cryptographic component. Finally, the *Adaptor component* implements the communication security policy by properly encrypting/decrypting messages. For that purpose, it uses the services offered by the Cryptographic component and the Rekeying component. The Adaptor component provides application components with the same interface as the Network Communication components. It is possible to transparently insert and remove the whole Security

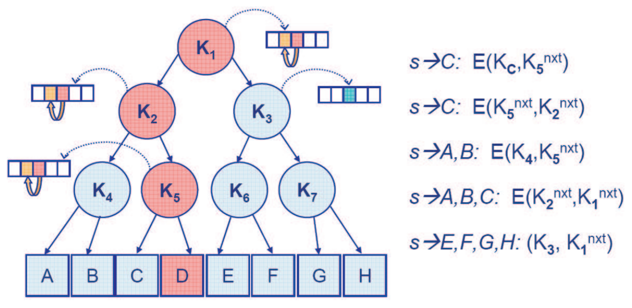


Fig. 10. The rekeying tree.

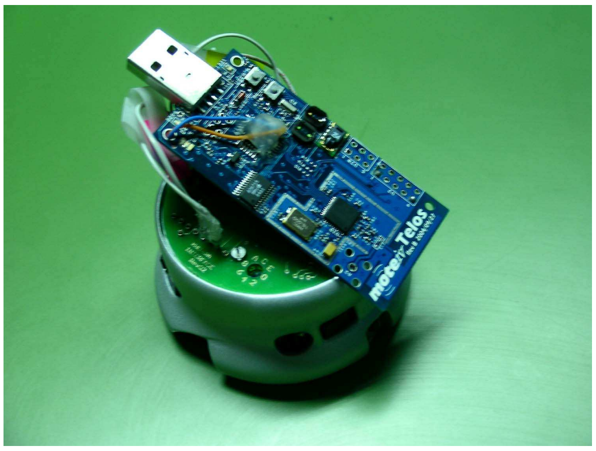


Fig. 11. Khepera robot with a Telos mote mounted on top.

Component Framework without affecting the functionality of the other components.

The Rekeying component guarantees forward security by implementing a scalable rekeying protocol, which refreshes the group key whenever a node leaves the group. The rekeying protocol scales to a large number of nodes because its communication overhead is logarithmic, and the computing overhead is kept low by using only hash functions to verify the authenticity of newly deployed keys.

When a node leaves, a centralized Key Distribution Server is responsible for distributing the new group key to all nodes except the leaving one. These nodes have only to verify the freshness and the authenticity of the keys coming from the server. The key authentication mechanism levers on key-chains, a technique based on one-time passwords. A key-chain is a set of symmetric keys, such that each key is the hash pre-image of the previous one under a one-way hash function. Hence, given a key in the key-chain, anybody can compute all the previous keys, but nobody can compute any of the next keys. Keys are revealed in the reversed order with respect to creation. Given an authenticated key in the key-chain, the nodes can authenticate the next keys by simply applying a hash function.

In order to reduce the communication overhead, the server maintains a tree structure of keys, see Figure 10. The internal tree nodes are associated with key-chains, while each leaf is associated with a symmetric private key, which each group member secretly shares with the server. A group member stores the last-revealed key for every internal tree node belonging to the path from its leaf to the root. Hence, the key associated to the tree root is shared by all group members and it acts as the group-key. When a group member leaves the group, all its keys become compromised and have to be redistributed. For example, let us suppose that group member D in Figure 10 leaves. The server then has to securely broadcast a new key for each internal tree node whose subtree contains the D leaf (e.g., nodes numbered with 1, 2, and 5 in the figure). In case of a binary tree, the server has to broadcast $2\log(n)-1$ messages where n is the network size. A more detailed description of the rekeying protocol can be found in [20].

VI. Hardware and Software Integration

Demonstration of the component-based design approach through the network reconfiguration scenario requires integration of both hardware and software. In this section, these efforts are described.

A. Hardware integration

The main hardware components for demonstrating the physical network reconfiguration are mobile robots and a wireless sensor network. A heterogeneous set of mobile robots are used. The wireless sensor nodes are the Telos and Tmote Sky motes, which are low power IEEE 802.15.4 compliant wireless sensor modules [26]. The Telos motes are equipped with humidity, light and temperature sensors. As part of the experimental validation this sensor network was made to interact with numerous mobile robotic platforms.

Figure 11 shows the simplest configuration used in our experiments, a Telos mote mounted on top of a Khepera robot. In this instance, the robot and mote communicate through their serial ports. The Telos acts both as a sensor and a radio interface for the mobile robot. The platform shown in Figure 11 was used for the early development of the network reconfiguration component.

The localization component was developed and tested on an RBbot mobile robot, shown in Figure 12. The control computations are done both in an integrated Tmote Sky mote and in an AVR processor. An I^2C bus is used to expand computational capabilities and to allow data exchange between components and external hardware. Localization is done using a Tmote Sky motes with an ultrasound sender unit, as shown to the upper right in the figure. This mote is mounted at the top of the robot. A Tmote Sky mote with ultrasound receiver is shown to the lower right. Such motes are placed along the walls of the tunnel and communicate with the robots to provide position information.

The collision avoidance component was developed on a fleet of mobile robots like the ones shown in Figure 13. Each robot comprises one Tmote Sky that implements the wireless communication via 811.15.4 protocol, and part of the control

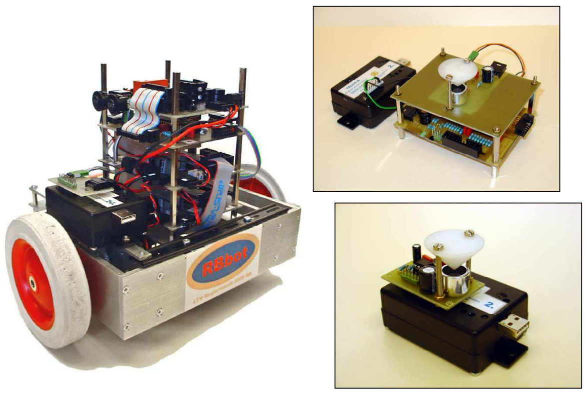


Fig. 12. RBbot robot and ultrasound equipped Tmote Sky.

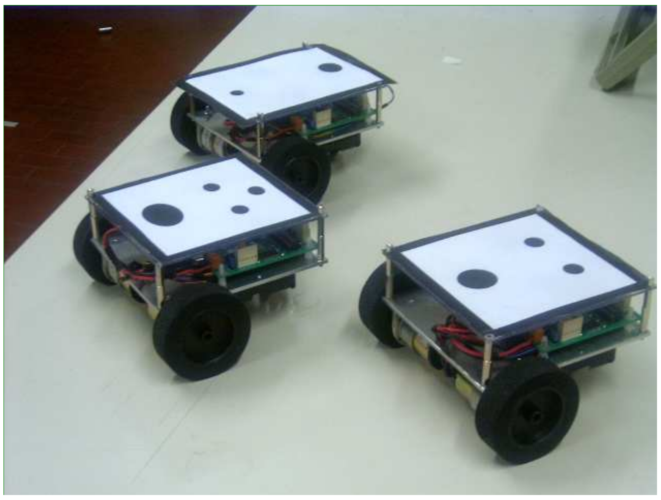


Fig. 13. Robots used for the development of the collision avoidance component.

algorithm. It also comprises three PSoC micro-controllers, connected via I^{2}C with one another. These micro-controllers perform the computations necessary to drive the robot. The mote and the micro-controllers exchange information (encoder readings, motors actuation, execution of rest of the control algorithm) via the RS232 serial interface.

The power control component was developed and tested for a wireless sensor network that consists of Telos motes. The transceiver of these nodes uses a Direct Sequence Spread Spectrum (DSSS) technique. Data are coded according to a DSSS operation, and then transmitted through a CSMA/CA technique. The Telos motes provide RSSI measurements defined as an average of the received signal power calculated over eight symbol periods. The implemented power control algorithms are based on these measurements and a new radio model that was developed for the Telos motes.

B. *Software integration*

The connecting theme underlying the different robotic platforms and the wireless sensor network is the component software. This is based on the Contiki operating system, which runs on all the mote platforms used in the experiments. The use of a common software substrate means that components developed by one team on one robotic platform can be ported to other platforms. The hardware resources available on the robot are exploited by the different components via the use of the same interface protocol.

Contiki is a lightweight and flexible operating system for tiny networked sensors [22]. It is built around a simple event-driven kernel on top of which application programs are written with stack-less threads. Thus, programs can be written in a threaded fashion, while interprocess communication is enabled using message passing through events. Contiki has a dynamic structure that allows to replace programs and drivers during runtime. Contiki provides also an implementation of a so called μIP stack for TCP and UDP communication [21]. Contiki implements μAODV, a light-weight implementation of the AODV ad-hoc routing protocol. AODV [28] stands for Ad-hoc On-Demand Distance Sensor Vector routing, which, contrary to most routing mechanisms, does not rely on periodic transmission of routing messages between the nodes. Instead, routes are created on-demand, i.e., only when actually needed to send traffic between a source and a destination node. This leads to a substantial decrease in the amount of network bandwidth consumed to establish routes.

The implementation of the component model for Contiki is called a component runtime kernel (CRTK). It allows the instantiation of a variety of components and the dynamic runtime binding of them. A component can be substituted with another component that has the same interface. Due to the memory limitations of the Telos motes, the dynamic run-time binding of the control components has not been explored in the demonstrations. The hardware limitations moreover forced the use of more powerful processors, such as AVR processors, to be connected with the Telos motes through I^{2}C buses, since the computational power of the motes was not sufficient to execute all the control components.

VII. Validation

This section describes a computer simulation of the scenario and some preliminary experimental results.

A. *TrueTime simulation*

In order to validate the network reconfiguration scenario, a simulation model has been developed. A holistic simulation approach is crucial for this, because it should be possible to simultaneously simulate the computations that take place within the nodes, the wireless communication between the nodes, the power devices (batteries) in the nodes, the sensor and actuator dynamics, the dynamics of the mobile robots, and the dynamics of the environment. In order to evaluate the limited resources correctly, the simulation model must be quite realistic. For example, it should be possible to simulate the computational delays associated with the execution of the software components. It should also be possible to simulate the effects of collisions and contention in the wireless medium access control (MAC) layer, the propagation of the ultrasound pulses, as well as the effects of the limited bandwidth of the communication bus used within the mobile robots.

There are a number of simulation environments available for networked control and sensor networks. However, the majority of these only simulate the wireless communication and the node computations. TrueTime [17], [13] is a co-simulation tool that has been developed at Lund University since 1999. By using TrueTime it is possible to simulate the temporal behavior of computer nodes and communication networks that interact with the physical environment. This makes it possible to concurrently simulate all the aspects described above.

TrueTime is a Matlab/Simulink-based tool that facilitates simulation of the temporal behavior of multi-tasking and event-based real-time kernels that execute controller tasks. The tasks are controlling physical systems, which are modeled as ordinary continuous-time Simulink blocks. TrueTime also makes it possible to simulate simple models of wired and wireless communication networks and their influence on networked control loops. The kernel block of TrueTime is event-driven and executes code that models, e.g., I/O tasks, control algorithms, and network interfaces. The scheduling policy of the individual kernel blocks is arbitrary and can be decided by the user. Likewise, in the network, messages are sent and received according to a chosen network model.

The TrueTime simulation of the tunnel scenario consists of two parts: a Simulink diagram containing the nodes, robots, and networks, and an 2D dynamic animation. The Simulink diagram is shown in Figure 14.

The stationary sensor nodes are implemented as Simulink subsystems that internally contain a TrueTime kernel modeling the Tmote Sky mote and connections to the radio network and the ultrasound communication blocks. In order to reduce the wiring, *From* and *To* blocks are used for the connections. The blocks handling the dynamic animation are not shown in the figure. The mobile robots, two RBbots as described in Section VI, are modeled as Simulink subsystems. Internally, these subsystems contain a TrueTime kernel modeling a Tmote Sky mote; a TrueTime kernel modeling an ATMEL AVR Mega16 processor, which acts as an interface to the ultrasound receiver and the proximity sensor used for obstacle detection; a TrueTime kernel modeling an ATMEL AVR Mega128 processor, which is used as a compute engine; two TrueTime kernels modeling two ATMEL AVR Mega16 processors, which are used as interfaces to the wheel motors; a model of the robot dynamics; and a subsystem representing the internal I^{2}C bus of the robot.

The ultrasound propagation is modeled by a separate network block, which is implemented in a similar fashion as the wireless network block. The components are implemented as TrueTime tasks and interrupt handlers. The wireless radio communication is modeled as the IEEE 802.15.4 protocol (the radio MAC protocol used in the Tmote Sky motes).

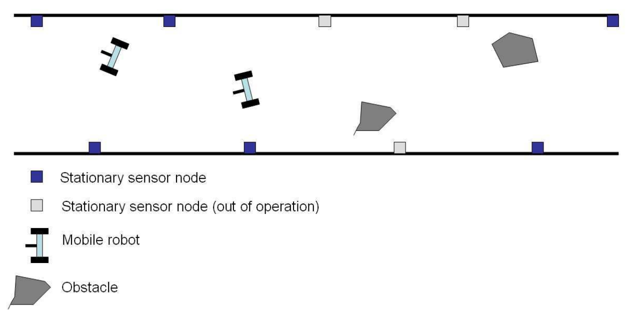


Fig. 15. Animation workspace for the TrueTime simulation.

The routing is implemented using a simulation model of the AODV protocol. The AODV protocol is in TrueTime implemented in such a way that it stores messages to destinations for which no valid route exists at the source node. This means that when, eventually, the network connectivity has been restored through the use of the mobile radio gateways, the communication traffic will be automatically restored.

The position of the robots and status of the stationary sensor nodes, i.e., whether they are operational or not, are shown in a separate animation workspace, see Figure 15.

The TrueTime simulation environment is further described in [15].

B. *Experimental validation*

The components outlined above were all implemented on the various robotic and sensor platforms and their performance was experimentally validated.

Figure 16 shows an example run of the collision avoidance component using three robots (see also Figure 13). The top sub-figure shows the origins and destinations of the three robots while the bottom two show the collision avoidance procedure with the reserved disks highlighted.

Figure 17 shows experiments for the development of the network reconfiguration component. Four robots and a base station (not pictured) are involved. All robots carry Telos motes. The two Kheperas (standing on boxes, see also Figure 11) are stationary in this experiment. The spider robot (in the background) is trying to maintain connectivity with the base station, multi-hopping its signals over any other available nodes if necessary. The transmission power of all motes is artificially reduced so that as the spider moves away from the base station it loses its connection with the rest of the network. The rover robot (in the middle of the picture) then moves into the gap between the spider and the network, to act as a relay node.

Figure 18 shows experiments for the development of the localization component. The robot shown in the forefront (see also Figure 12) is trying to navigate down the corridor based on localization information collected from ultrasound equipped Tmote Sky motes situated on either side near the walls. The two graphs show the improvement in the navigation if this localization information is used (right) vs. open loop navigation (left).

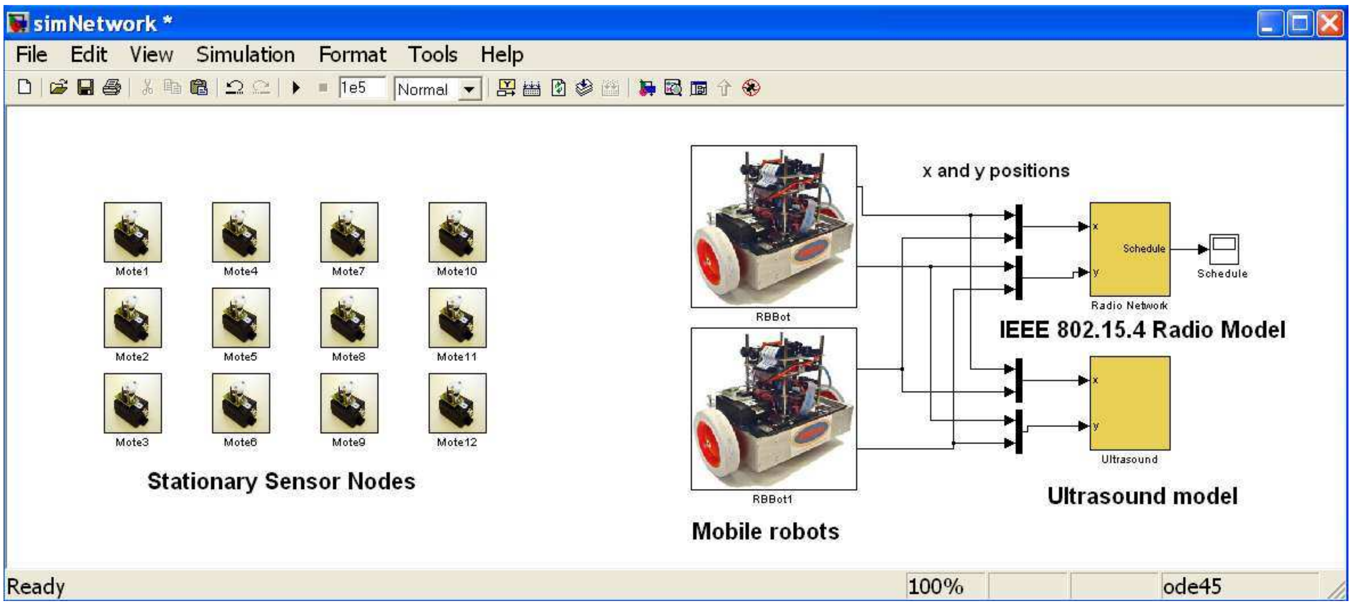


Fig. 14. The Simulink model diagram. In order to reduce the use of wires *From* and *To* blocks are used to connect the stationary sensor nodes to the radio and ultrasound networks.

Figure 19 shows experimental results with the power control component. The temporal evolution of the averaged RSSI is presented for a situation when three transmitting network nodes are connected to the receiver node mounted on the robot. The power is adjusted according to the MIAD power control algorithm discussed in Section IV. Node 1 is not within line-of-sight, whereas nodes 2 and 3 are. Therefore, the signal strength of links 2 and 3 settle quickly to their appropriate fixed values, while link 1 oscillates in accordance with the MIAD control strategy.

Finally, Figure 20 shows a model of a road tunnel developed for the demonstration of the disaster relief scenario. Trucks, cars and firefighters are indicated by lights in the tunnel. The position of the Telos mote shows where the wireless nodes are positioned in this particular demonstration.

VIII. Concluding remarks

We outlined techniques at the heart of the design and control of complex networked embedded systems. Even though the work presented was motivated by a specific tunnel disaster relief scenario, we believe that the control components developed provide a suitable framework for addressing control problems in a wide range of applications; possible target applications include surveillance and environmental monitoring, critical infrastructure protection, transportation, agriculture, industrial automation etc. The research presented here bridges the gap between control, communication and computation technologies and suggests a number of productive and interesting research directions.

Acknowledgments

The work described in this paper has been done within the European project RUNES, FP6-IST-004536. It has been a major effort of Workpackage 6 of this project and several people have contributed to it. Among them P. Alriksson, D. Asztalos, R. Braun, A. Cervin, A. Danesi, A. Fagiolini, C. Fischione, D. Henriksson, M. Johansson, E. Kolybas, I. Koveos, G. Nikolakopoulos, J. Nordh, M. Ohlin, L. Pallottino, A. Panousopoulou, P. G. Park, I. M. Savino, R. Schiavi, A. Speranzon, G. Terzakis, B. Zurita Ares. Their contrinbutions are gratefully acknowledged.

References

[1] *Automotive Open System Architecture (AUTOSAR)*. http://www.autosar.org.

[2] *Common Object Request Broker Architecture (CORBA)*. http://www.omg.org/gettingstarted/corbafaq.htm.

[3] *CORBA Component Model Specification*. http://www.omg.org/technology/documents/formal/components.htm.

[4] *Enterprise JavaBeans Technology*. http://java.sun.com/products/ejb.

[5] *Java Remote Method Invocation (RMI)*. http://java.sun.com/javase/technologies/core/basic/rmi.

[6] *Microsoft Component Object Model (COM)*. http://www.microsoft.com/com.

[7] *Microsoft .NET*. http://www.microsoft.com/net.

[8] *Reconfigurable Ubiquitous Networked Embedded Systems*. http://www.ist-runes.org.

[9] D5.1—Survey of middleware for networked embedded systems. RUNES deliverable, http://www.ist-runes.org/public_deliverables.html, 2005.

[10] D5.2—RUNES middleware architecture. RUNES deliverable, http://www.ist-runes.org/public_deliverables.html, 2005.

[11] A. Sampath and P.S. Kumar and J.M. Holtzman. On setting reverse link target sir in a cdma systems. In *Proc. of IEEE VTC*, 1997.

[12] P. Alriksson, J. Nordh, K.-E. Årzén, A. Bicchi, A. Danesi, R. Schiavi, and L. Pallottino. A component-based approach to localization and collision avoidance for mobile multi-agent systems. In *European Control Conference*, Kos, Greece, 2007.

[13] M. Andersson, D. Henriksson, A. Cervin, and K.-E. Årzén. Simulation of wireless networked control systems. In *Proceedings of the 44th IEEE Conference on Decision and Control and European Control Conference ECC 2005*, Seville, Spain, December 2005.

[14] B. Zurita Ares, C. Fischione, A. Speranzon, and K. H. Johansson. On power control for wireless sensor networks: System model, middleware component and experimental evaluation. In *European Control Conference*, Kos, Greece, 2007.

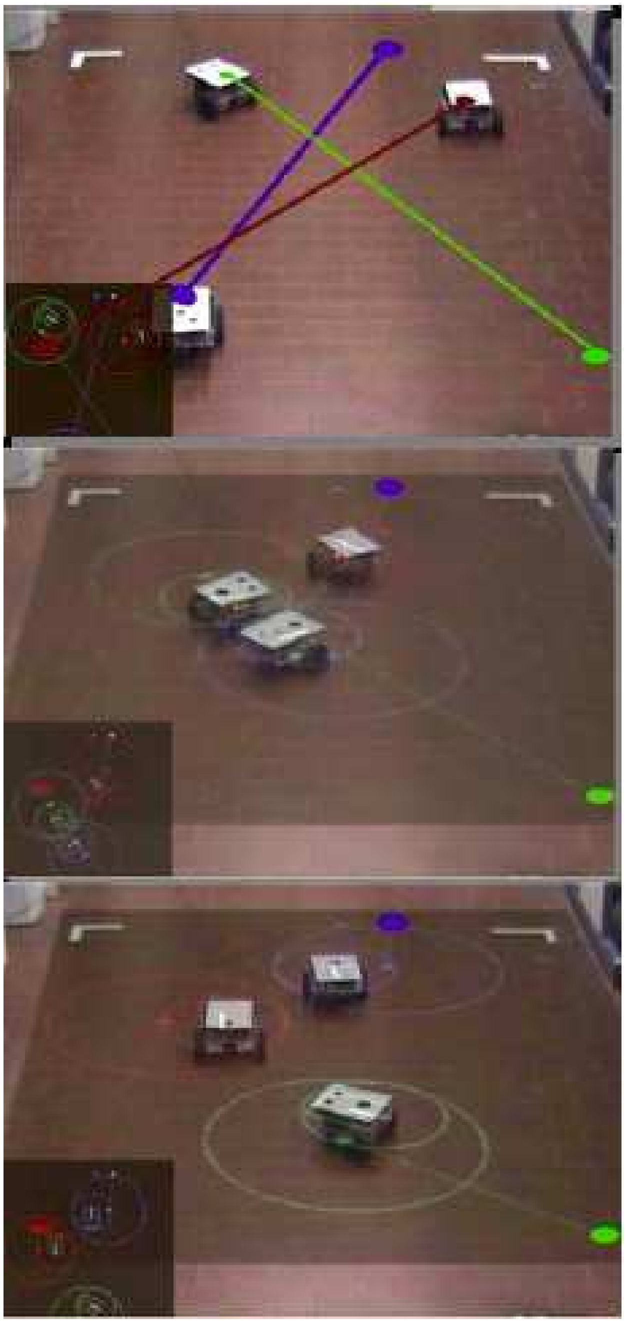


Fig. 16. Example of collision avoidance component experimental results.

[15] K.-E. Årzén, M. Ohlin, A. Cervin, P. Alriksson, and D. Henriksson. Holistic simulation of mobile robot and sensor network applications using TrueTime. In *European Control Conference*, Kos, Greece, 2007.

[16] G. Baliga, S. Graham, L. Sha, and P. R. Kumar. Service continuity in networked control using Etherware. *IEEE Distributed Systems Online*, 5(8), 2004.

[17] A. Cervin, D. Henriksson, B. Lincoln, J. Eker, and K.-E. Årzén. How does control timing affect performance? *IEEE Control Systems Magazine*, 23(3):16–30, June 2003.

[18] A. Deshpande, A. Gollu, and L. Semenzato. The SHIFT programming language for dynamic networks of hybrid automata. *IEEE Transactions on Automatic Control*, 43(4):584–587, April 1998.

[19] L. Di Paolo, C. Fischione, F. Graziosi, F. Santucci, and S. Tennina. Performance analysis of distributed source coding and packet aggregation in wireless sensor networks. In *IEEE Globecom*, 2006.

[20] G. Dini and I. M. Savino. S2rp: a secure and scalable rekeying protocol

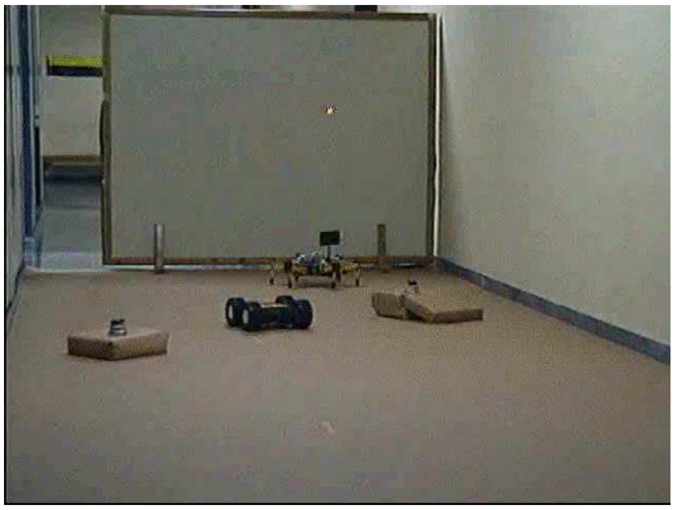
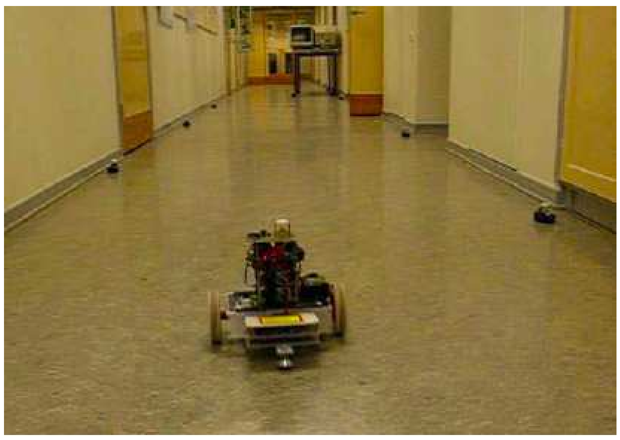


Fig. 17. Example of network reconfiguration component experimental results.



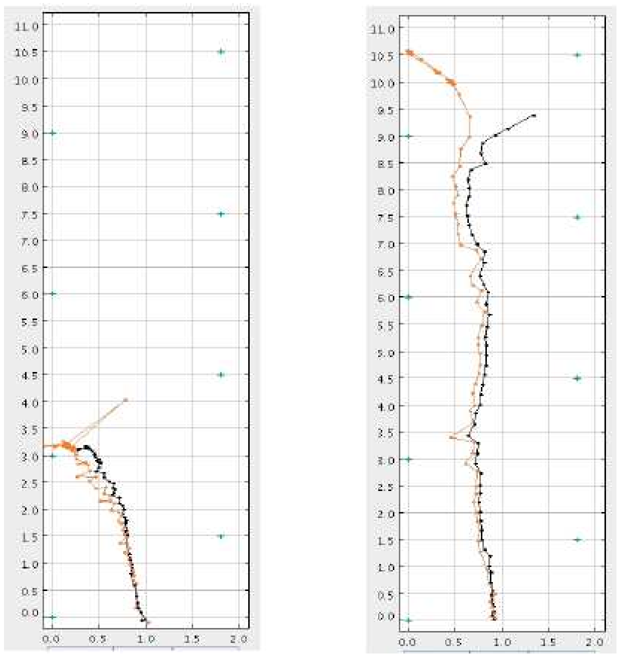


Fig. 18. Example of localization component experimental results.

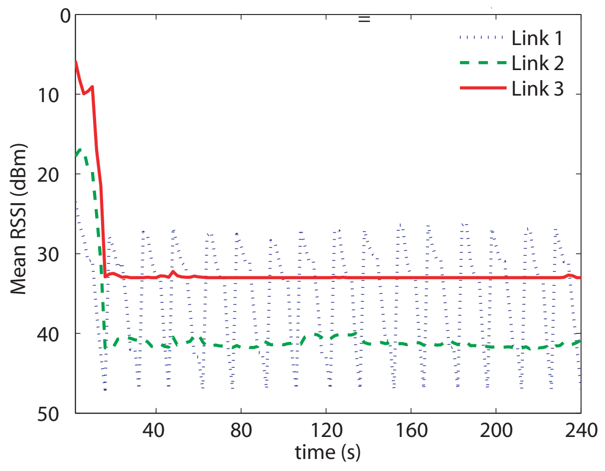


Fig. 19. Example of power control component experimental results. Temporal evolution of the RSSI for an implementation of the MIAD power control algorithm.

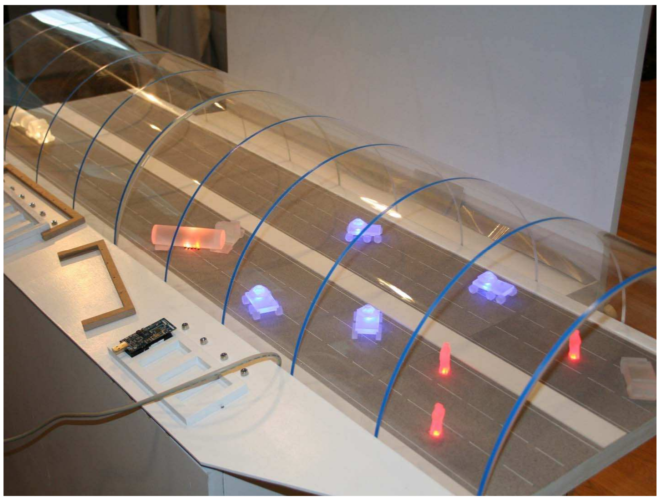


Fig. 20. Road tunnel model demonstrator.

for wireless sensor networks. In *The Third IEEE International Conference on Mobile Ad-hoc and Sensor Systems (MASS'06)*, Vancouver, Canada, October 9–12 2006.

[21] A. Dunkels, J. Alonso, and T. Voigt. Making TCP/IP viable for wireless sensor networks. In *Proceedings of the First European Workshop on Wireless Sensor Networks*, 2004.

[22] A. Dunkels, B. Gröonvall, and T. Voigt. Contiki—a lightweight and flexible operating system for tiny networked sensors. In *Proceedings of IEEE Workshop on Embedded Networked Sensors*, 2004.

[23] B. S. Heck, L. M. Wills, and G. J. Vachtsevanos. Software technology for implementing reusable, distributed control systems. *IEEE Control Systems Magazine*, 23(1):21–35, 2003.

[24] J. Chou, D. Petrovic, K. Ramchandran. A distributed and adaptive signal processing approach to reducing energy consumption in sensor networks. In *IEEE INFOCOM*, 2003.

[25] H. Kopetz and G. Bauer. The time-triggered architecture. *Proc. of the IEEE*, 91:112–126, 2003.

[26] Moteiv Corporation, http://www.moteiv.com. *Telos: Ultra low power IEEE 802.15.4 compliant wireless sensor module*, 2006.

[27] A. Panousopoulou and A. Tzes. Utilization of mobile agents for voronoi-based heterogeneous wireless sensor network reconfiguration. In *European Control Conference*, Kos, Greece, 2007.

[28] C.E. Perkins and E.M. Royer. Ad-hoc on-demand distance vector (AODV) routing. In *Proceedings of the 2nd IEEE Workshop on Mobile Computing Systems and Applications*, New Orleans, LA, 1999.

[29] S. Sastry, J. Sztipanovits, R. Bajcsy, and H. Gill (Editors). Special issue on modeling and design of embedded software. *Proceedings of the IEEE*, 91(1), 2003.

[30] D. Schmidt and F. Kuhns. An overview of the real-time CORBA specification. *IEEE Computer*, 2000.

[31] R. Zhang, C. Lu, T. F. Abdelzaher, and J. A. Stankovic. Control-Ware: A middleware architecture for feedback control of software performance. In *International Conference on Distributed Computing Systems*, Vienna, Austria, 2002.