Holistic Simulation of Mobile Robot and Sensor Network Applications Using TrueTime

Årzén, Karl-Erik; Ohlin, Martin; Cervin, Anton; Alriksson, Peter; Henriksson, Dan

2007

Holistic Simulation of Mobile Robot and Sensor Network Applications Using TrueTime

K.-E. Årzén, M. Ohlin, A. Cervin, P. Alriksson, D. Henriksson
Department of Automatic Control LTH
Lund University, Sweden

Abstract— **The RUNES project defines a complex road tunnel scenario involving multiple mobile robots navigating in a sensor network environment. In this paper, a TrueTime simulation model of the tunnel scenario is developed. The TrueTime simulator allows concurrent simulation of the physical robots and their environment, the software in the nodes, the radio communication, the network routing, and the ultra-sound navigation system.**

I. Introduction

Sensor/actuator networks and mobile robots are application areas for embedded real-time systems where wireless communication plays a vital role. The computing and communication resources are often severely limited, making integrated design approaches important. Another common characteristic is that the systems interact with their environment. One example is a sensor network that monitors the presence of moving objects in the environment. Other examples are mobile robots moving around in the environment or sensor/actuator networks that implement networked control loops.

Within the EU/IST FP6 Integrated Project RUNES (Reconfigurable Ubiquitous Networked Embedded Systems) a disaster relief road tunnel scenario has been defined. In the scenario mobile robots are used as mobile radio gateways that ensure the connectivity of a sensor network located in a road tunnel in which an accident has occurred. The RUNES tunnel scenario is described in more detail in the companion paper [1]. A number of software components have been developed for this scenario. These are described in the companion papers in this session. A localization component based on ultrasound is used for localizing the mobile robots and a collision avoidance component ensures that the robots do not collide, see [2] for a description of both these components. A network reconfiguration component [3] and a power control component [4] are responsible for deciding the best position for the mobile robot in order to maximize radio connectivity, and for adjusting the radio power transmit level [4].

In parallel with the actual implementation of this scenario a simulated version is being developed. The focus of the simulation is the timing aspects of the scenario. Things that are of interest to evaluate in simulation include the execution time of the software in the stationary sensor network nodes and in the mobile robots, the dynamics of the mobile robots, the utilization of the wireless communication media, and the propagation time of the ultrasound used in the localization of the mobile robots.

Simulation is a powerful technique that can be used at several stages of system development. In order to support the application at hand, an holistic simulation approach is crucial. 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, and the dynamics of the mobile robots. In order to model the limited resources correctly, the simulation model must be quite realistic. For example, it should be possible to simulate the timing effects of interrupt handling in the microcontrollers implementing the control logic of the nodes. It should also be possible to simulate the effects of collisions and contention in the wireless communication. Due to simulation time and size constraints, it is at the same time important that the simulation model is not too detailed. For example, simulating the computations on a source code level, instruction for instruction, would be overly costly. The same applies to simulation of the wireless communication at the radio interface level or on the bit transmission level.

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. Hence, something more is needed. TrueTime [5], [6] is a MATLAB/Simulink-based cosimulation tool that has been developed at Lund University since 1999. Using TrueTime, it is possible to concurrently simulate all the aspects mentioned above.

TrueTime provides a small but powerful block library, see Fig. 1. The kernel block executes code that models, e.g., I/O tasks, control algorithms, and network interface drivers. 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. TrueTime is available for download at `http://www.control.lth.se/truetime/`.

TrueTime can be used as an experimental platform for research on dynamic real-time control systems. For instance, it is possible to study compensation schemes that adjust the control algorithm based on measurements of actual timing variations (i.e., to treat the temporal uncertainty as a disturbance and manage it with feedforward or gain scheduling). It is also easy to experiment with more flexible approaches to real-time scheduling of controllers, such as feedback scheduling, see [7], or as is the case in this paper, to simulate MANET and sensor network applications.

The aim of this paper is to describe how TrueTime can be

ISBN: 978-960-89028-5-5

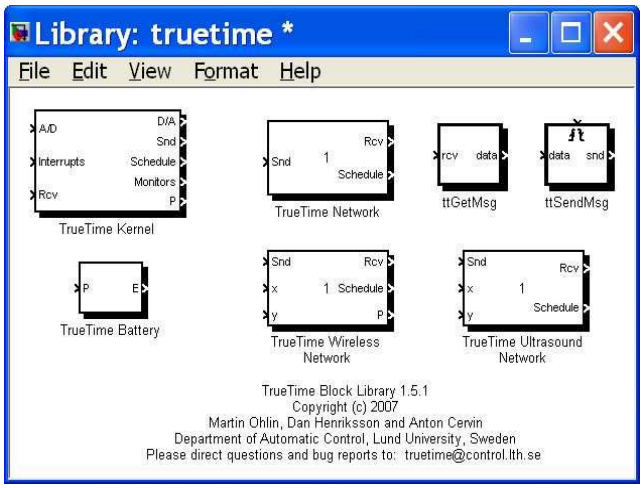

Fig. 1. The TrueTime block library.

used to model different aspects of these types of scenarios.

A. Outline of the Paper

The hardware used in the physical scenario is presented in Section II. Section III describes how TrueTime is used to model computers and computations. The mobile robots internally use an I^2C bus for communication. How this is modeled in TrueTime using the network block is discussed in Section IV. The radio communication and how it is modeled in TrueTime is discussed in Section V. A special ultrasound block has been developed to model the propagation of the ultrasound used by the localization component. This is described in Section VI. The ad hoc routing in the scenario is based on the AODV protocol. The TrueTime implementation of this is discussed in Section VII. The total simulation model including the sensor and actuator models are presented in Section VIII. Finally, related work is presented in Section IX.

II. The Physical Scenario Hardware

The physical scenario consists of a number of hardware and software components. The functionality of the software components is described in more detail in the companion papers in this session.

The hardware consists of the stationary wireless communication nodes and the mobile robots. The wireless communication nodes are implemented by Tmote Sky sensor network motes executing the Contiki operating system [8]. In addition to the ordinary sensors for temperature, light and humidity an ultrasound receiver has been added to each mote, see Fig. 2.

The mobile robots used vary among the partners. In this paper only the Lund robots, called RBbots, will be considered. The two RBbots used are shown in Fig. 3. Both robots are equipped with an ultrasound transmitter board (at the top). The robot to the left has the obstacle detection sensors mounted. This consists of a touch sensor bar and an IR proximity sensor mounted on an RC-servo that sweeps a circle segment in front of the robot.

Fig. 2. Stationary sensor network nodes with ultrasound receiver circuit. The node is packaged in a plastic box to reduce wear.

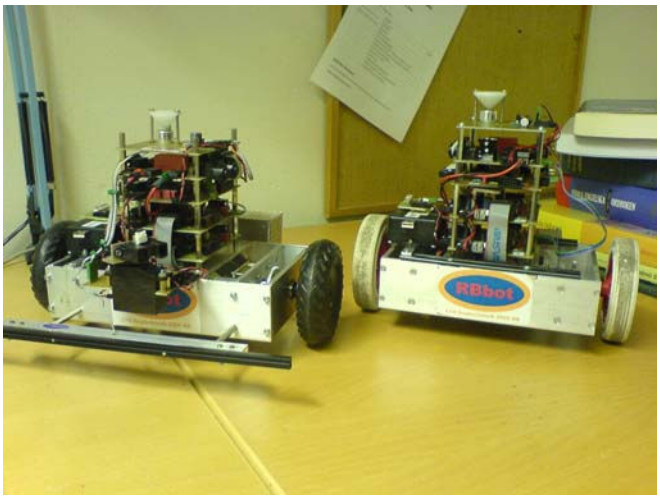

Fig. 3. The two Lund RBbots.

The RBbots internally consists of one Tmote Sky, one ATMEL AVR Mega128, and three ATMEL AVR Mega16 microprocessors. The nodes communicate internally over an I^2C bus. The Tmote Sky is used for the radio communication as the master. Two of the ATMEL AVR Mega16 processors are used as interfaces to the wheel motors and the wheel encoders measuring the wheel angular velocities. The third ATMEL AVR Mega16 is used as the interface to the ultrasound transmitter and the obstacle detection sensors. The AVR Mega128 is used as a compute engine for software component code that does not fit the limited memory of the TMote Sky. The structure is shown in Fig. 4.

III. TrueTime Modeling of Computations

Computers and computations are modeled in TrueTime by the kernel block. This is used to model the Tmote Sky TI MSP430 processors and the ATMEL AVR processors.

The kernel block is a Simulink S-function that simulates a computer with a real-time kernel, A/D and D/A converters, a network interface, and external interrupt channels. The kernel executes user-defined tasks and interrupt handlers. Internally, the kernel maintains several data structures that are commonly found in a real-time kernel: a ready queue, a time queue, and records for tasks, interrupt handlers, monitors and timers that have been created for the simulation.

An arbitrary number of tasks can be created to run in the TrueTime kernel. Tasks may also be created dynamically as the simulation progresses. Tasks are used to simulate both periodic activities, such as controller and I/O tasks,

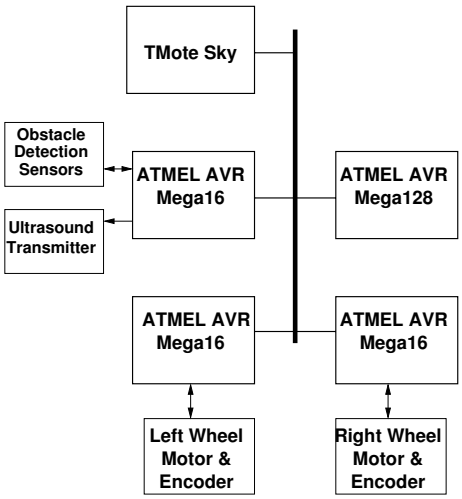

Fig. 4. RBbot hardware architecture.

```
function [exectime,data] = ctrl_code(segment,data)
switch segment,
  case 1,
    data.y = ttAnalogIn(1);
    data.u = calculate_output(data.x,data.y);
    exectime = 0.002;
  case 2,
    ttAnalogOut(1,data.u);
    data.x = update_state(data.x,data.y);
    exectime = 0.006;
  case 3,
    exectime = -1; % finished
end
```

Fig. 5. Example of a standard controller code function written in MATLAB code. The local memory of the control task is represented by the data structure `data`. This stores the input, the controller state, and the output between invocations of the code segments.

and aperiodic activities, such as communication tasks and event-driven controllers. Aperiodic tasks are executed by the creation of task instances (jobs). Each task is characterized by a number of static (e.g., relative deadline, period, and priority) and dynamic (e.g., absolute deadline and release time) attributes.

Interrupts may be generated in two ways: externally (associated with the external interrupt channel of the kernel block) or internally (triggered by user-defined timers). When an external or internal interrupt occurs, a user-defined interrupt handler is scheduled to serve the interrupt.

The execution of tasks and interrupt handlers is specified by user-written code functions. Algorithms may also be defined graphically using ordinary discrete Simulink block diagrams. Simulated execution occurs at three distinct priority levels: the interrupt level (highest priority), the kernel level, and the task level (lowest priority). The execution may be preemptive or non-preemptive; this can be specified individually for each task and interrupt handler.

A. Code Functions

The functionality of each task or interrupt handler is defined by a code function written in MATLAB or C++ code. The code function is divided into a number of segments, which are (normally) executed sequentially as the simulation progresses. Computational delay is simulated by associating with each code segment an execution time. The code segment is executed in zero simulation time. This is followed by a delay equal to the specified execution time before the next segment is executed. The delay may be preempted by higher-priority tasks or interrupt handlers, making the total simulation time between two segments greater than or equal to the execution time of the segment. The code may cause a task to self-suspend by calling certain kernel primitives. In this case, no further code segments are executed until the task is unblocked.

The code function format is quite flexible and allows the user to simulate loops and branches, input-output latencies, blocking when accessing shared resources, etc. The number of segments can be chosen in relation to the desired time granularity of the simulation. Technically it would, e.g., be possible to simulate very fine-grained details occurring at the machine instruction level, such as race conditions. However, that would require a very large number of code segments.

The listing in Fig. 5 shows an example of a code function implementing a standard regulator in state-space form. In the first segment, the plant is sampled and the control signal is computed (`calculate_output`). In the second segment, the control signal is actuated and the internal state is updated (`update_state`). The third segment indicates the end of execution by returning a negative execution time.

The data structure `data` represents the local memory of the task and is used to store the control signal and measured variable between calls to the different segments. A/D and D/A conversion is performed using the kernel primitives `ttAnalogIn` and `ttAnalogOut`.

The simulated execution time of each segment is returned by the code function, and can be modeled as constant, random, or data-dependent. Note that the input-output latency of this controller will be *at least* 2 ms (i.e., the execution time of the first segment). However, if there is preemption from other high-priority tasks or interrupt handlers, the actual input-output latency will be longer.

B. Synchronization

Synchronization between tasks is supported by semaphores, monitors with condition variables (events), and mailboxes. Monitors are used to guarantee mutual exclusion between tasks when accessing common data. Tasks waiting for monitor access are sorted by priority in the waiting queue. Basic priority inheritance is implemented as resource access protocol.

Events can be associated with monitors to represent condition variables. Events may also be free (i.e., not associated with a monitor). This feature can be used to obtain synchronization between tasks where no conditions on shared data are involved. As for monitors, the waiting queues of the free events are sorted after task priority. The mailboxes support both blocking and non-blocking read and write operations.

C. Scenario Hardware Models

The basic programming model used for the TI MSP430 processor used in the Tmote Sky systems is event-driven programming with interrupt handlers for handling timer interrupts, bus interrupts, etc. In TrueTime the same architecture can be used. However, the Contiki OS also supports so called protothreads [9]. Protothreads are lightweight stackless threads designed for severely memory constrained systems. Protothreads provide linear code execution for event-driven systems implemented in C. Protothreads can be used to provide blocking event-handlers. They provide sequential flow of control without complex state machines or full multi-threading. In TrueTime protothreads are modeled as ordinary tasks. The ATMEL AVR processors are modeled as event-driven systems. A single non-terminating task acts as the main program and the event-handling is performed in interrupt handlers.

The software executing in the TrueTime processors is written in C++. The names of the files containing the code are input parameters of the kernel blocks. The localization component consists of two parts, as described in the companion paper [2]. The distance sensor part of the components is implemented as a (proto-)thread in each stationary sensor node. The Extended Kalman Filter-based data fusion is implemented in the Tmote Sky processor onboard each robot. The localization method makes use of the ultrasound network and the radio network. The collision avoidance component code, also described in more detail in [2], is implemented in the ATMEL AVR Mega128 processor using events and interrupts. It interacts over the I^2C bus with the localization component and with the robot position controller, both located in the Tmote Sky processor.

IV. TrueTime Modeling of Bus Communication

The I^2C bus within the RBbots is modeled in TrueTime by a network block. The network block is event-driven and executes when messages enter or leave the network. When a node tries to transmit a message, a triggering signal is sent to the network block on the corresponding input channel. When the simulated transmission of the message is finished, the network block sends a new triggering signal on the output channel corresponding to the receiving node. The transmitted message is put in a buffer at the receiving computer node.

A message contains information about the sending and the receiving computer node, arbitrary user data (typically measurement signals or control signals), the length of the message, and optional real-time attributes such as a priority or a deadline.

The network block simulates medium access and packet transmission in a local area network. Six simple network models are currently supported: CSMA/CD (e.g., Ethernet), CSMA/AMP (e.g., CAN), Round Robin (e.g., Token Bus), FDMA, TDMA (e.g., TTP), and Switched Ethernet. The propagation delay is ignored, since it is typically very small in a local area network. Higher network layer protocols such as TCP can be implemented as user applications in the kernel blocks. Configuring the network blocks involve specifying a number of general parameters, such as transmission rate, network model, and probability for packet loss. Protocol-specific parameters that need to be supplied include the time slot and cyclic schedule in the case of TDMA.

The TrueTime network model assumes the presence of a network interface card or a bus controller implemented either in hardware or software, i.e. as drivers. The Contiki interface to the I^2C bus is software-based and corresponds well to the TrueTime model. In the ATMEL AVRs, however, it is normally the responsibility of the application programmer to manage all bus access and synchronization directly in the application code. In the TrueTime model this low-level bus access is not modeled. Instead it is assumed that there exists a hardware or software bus interface that implements this.

Although the I^2C is a multi-master bus that uses arbitration to resolve conflicts this is not how it is modeled in TrueTime. On the Tmote Sky the radio chip and the I^2C bus share connection pins. Due to this it is only possible to have one master on the I^2C bus and this master must be the Tmote Sky. All communication must be initiated by the master. Due to this bus access conflicts are eliminated. Therefore the I^2C bus is modeled as a CAN bus with the transmission rate set to match the transmission rate of the I^2C bus.

V. TrueTime Modeling of Radio Communication

The radio communication used by the Tmote Sky is the IEEE 802.15.4 MAC protocol (the so called Zigbee MAC protocol). TrueTime supports wireless radio communication through the wireless network blocks. Two wireless protocols are supported: IEEE 802.11 b/g (WLAN) and IEEE 802.15.4. The simulation covers the medium access delay and the packet transmission.

Both the WLAN and the Zigbee models share the following properties:

- Ad-hoc wireless networks, as opposed to infrastructure-based ones.
- Isotropic antenna.
- Unable to send and receive at the same time.
- Path loss of radio signals modeled as $\frac{1}{d^a}$ where d is the distance in meters and a is a suitably chosen parameter to model the environment.
- The possibility for the user to define his own path loss model to, e.g., take multi-path propagation and fading into account. The model is represented by a user-defined Matlab-function.
- Interference from other terminals (shared medium).
- ACK messages on the MAC protocol level.

In the $802.15.4$ case, a package transmission is modeled like this: The node that wants to transmit a packet starts by waiting a random number of backoff periods. The message is then transmitted if the medium is idle. If the medium is not idle, the window in which the delay is chosen is increased and the node waits for a random number of backoff periods again. This behaviour continues until the message is either transmitted or the maximum number of retries is reached. When a node starts to transmit, its relative position to all other nodes in the same network is calculated, and the signal level in all those nodes are calculated according to the path-loss formula $\frac{1}{d^a}$ or the user-defined path-loss formula.

The signal is assumed to be possible to detect, if the signal level in the receiving node is larger than a configurable threshold (receiver signal threshold). If this is the case, then the signal-to-noise ratio (SNR) is calculated and used to statistically find the block error rate (BLER). Note that all other transmissions add to the background noise when calculating the SNR. The BLER, together with the size of the message, is used to calculate the number of bit errors in the message and if this number is lower than another threshold (error coding threshold), then it is assumed that the channel coding scheme is able to fully reconstruct the message. If there are (already) ongoing transmissions from other nodes to the receiving node and their respective SNRs are lower than the new one, then all those messages are marked as collided. Also, if there are other ongoing transmissions which the currently sending node reaches with its transmission, then those messages may be marked as collided as well.

Note that a sending node does not know if its message is colliding, therefore ACK messages are sent on the MAC protocol layer. From the perspective of the sending node, lost messages and message collisions are the same, i.e., no ACK is received. If no ACK is received during a certain configurable time, the message is retransmitted according to the same scheme as described above. There are only a certain configurable number of retransmissions before the sender gives up on the message and it is not retransmitted anymore.

A. Network Reconfiguration and Radio Power Control

The requirements on the simulation environment from the network reconfiguration and radio power control components are that it should be possible to change the transmit power of the nodes and that it should be possible to measure the received signal strength, i.e., the so called Received Signal Strength Indicator (RSSI). The former is possible through the TrueTime command

```
ttSetNetworkParameter('transmitpower',value)
```

The RSSI is obtained as an optional return value of the TrueTime function `ttGetMsg`, which is defined as

```
[msg, signalPower] = ttGetMsg(network)
```

This function is typically called from the interrupt handler associated with the network to extract the received message, `msg`.

VI. TrueTime Ultrasound Model

In order to model the ultrasound a special block has been developed. The block is a special version of the wireless network block that models the ultrasound propagation of a transmitted ultrasound pulse. Senders and receivers are connected to the block through the send and receive ports, similar to an ordinary wireless network block. The x and y positions of the senders and receivers are also inputs to the block.

The main difference between the wireless network block and the ultrasound block is that in the ultrasound block it is the propagation delay that is important, whereas in the ordinary wireless block it is the medium access delay and the transmission delay that are modeled. The ultrasound is modeled as a single sound pulse. When it arrives at a stationary sensor node an interrupt is generated. This also differs from the physical scenario, in which the ultrasound signal is connected via an AD converter to the Tmote Sky.

VII. Routing

The network routing is implemented using a TrueTime model of the AODV protocol. AODV [10] stands for Ad-hoc On-Demand Distance Vector routing and contrary to most routing mechanisms, it 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.

AODV uses three basic types of control messages in order to build and invalidate routes: route request (RREQ), route reply (RREP), and route error (RERR) messages. These control messages contain source and destination sequence numbers, which are used to ensure fresh and loop-free routes.

A node that requires a route to a destination node initiates route discovery by broadcasting an RREQ message to its neighbors. A node receiving an RREQ starts by updating its routing information backwards towards the source. If the same RREQ has not been received before, the node then checks its routing table for a route to the destination. If a route exists with a sequence number greater than or equal to that contained in the RREQ, an RREP message is sent back towards the source. Otherwise, the node rebroadcasts the RREQ. When an RREP has propagated back to the original source node, the established route may be used to send data. Periodic hello messages are used to maintain local connectivity information between neighboring nodes. A node that detects a link break will check its routing table to find all routes which use the broken link as the next hop. In order to propagate the information about the broken link, an RERR message is then sent to each node that constitute a previous hop on any of these routes.

Two TrueTime tasks are created in each node to handle AODV send and receive actions, respectively. The AODV send task is activated from the application code as a data message should be sent to another node in the network. The AODV receive task handles incoming AODV control messages and forwarding of data messages. Communication between the application layer and the AODV layer is handled using TrueTime mailboxes. Each node also contains a periodic task, responsible for broadcasting hello messages and determine local connectivity based on hello messages received from neighboring nodes. Finally, each node has a task to handle timer expiry of route entries.

The AODV protocol in TrueTime is 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.

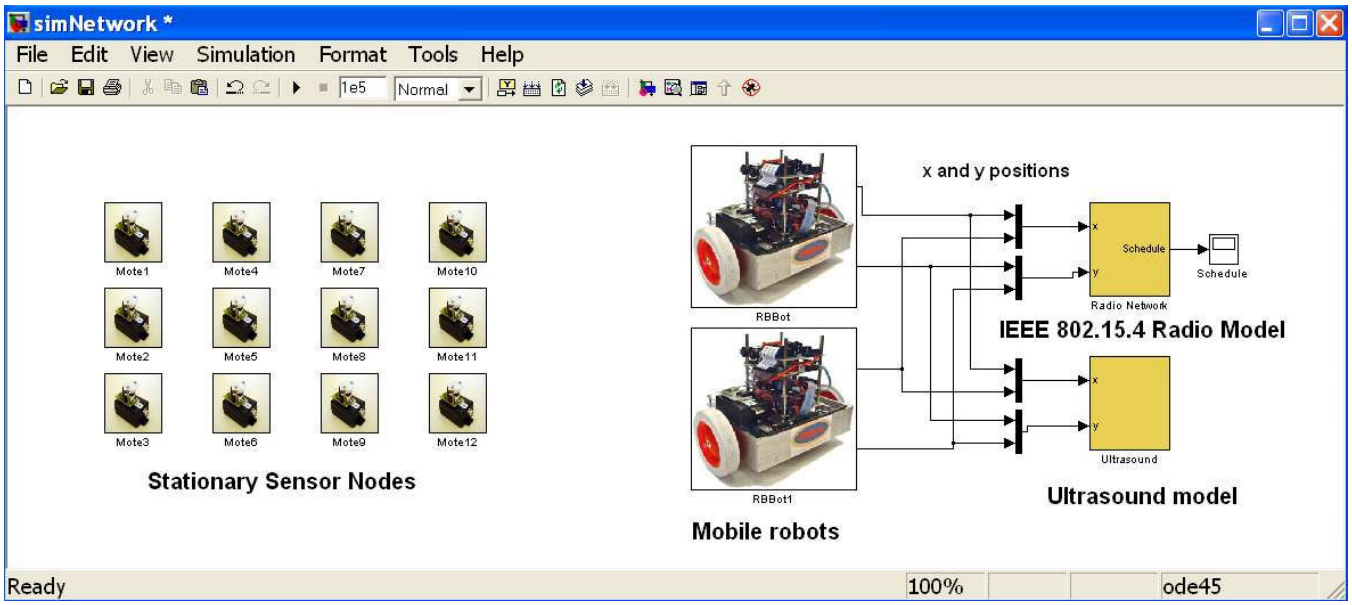

Fig. 6. The TrueTime model diagram. In order to reduce the use of wires From and To blocks hidden inside the corresponding subsystems are used to connect the stationary sensor nodes to the radio and ultrasound networks.

VIII. The Complete Model

In addition to the above the complete model for the scenario also contains models of the sensors, motors, robot dynamics, and a world model that keeps track of the position of the robots and the fixed obstacles within the tunnel.

The wheel motors are modeled as first-order linear systems plus integrators with the angular velocities and positions as the outputs. From the motor velocities the corrsponding wheel velocities are calculated. The wheel positions are controlled by two PI-controllers residing in the ATMEL AVR processors acting as interfaces to the wheel motors.

The Lund RBbot is a dual-drive unicycle robot. It is modeled as a third-order system

$$\begin{aligned} \dot{p}_x &= \frac{1}{2}(R_1\omega_1 + R_2\omega_2)\cos(\theta) \\ \dot{p}_y &= \frac{1}{2}(R_1\omega_1 + R_2\omega_2)\sin(\theta) \\ \dot{\theta} &= \frac{1}{D}(R_2\omega_2 - R_1\omega_1) \end{aligned} \tag{1}$$

where the state consists of the x- and y-positions and the heading θ. Input to the system are the angular velocities ω_1 and ω_2 of the two wheels. The parameters R_1 and R_2 are the radius of the two wheels and D is the distance between the wheels.

The top-level TrueTime model diagram is shown in Fig. 6. 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 hidden inside the corresponding subsystems are used for the connections. The block handling the dynamic animation is not shown in the figure.

The subsystem for the mobile robots is shown in Fig. 7. The Robot Dynamics block contains the motor models and the robot dynamics model.

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 Fig. 8.

IX. Limitations of the Model

The implemented TrueTime model contains several simplifications. For example, interrupt latencies are not simulated, only context switch overheads. All execution times are chosen based on experience from the hardware implementation. The component framework is not implemented as such. However, since most of the component activities, e.g., composition, are performed off-line this is not a major limitation. Also, it is important to stress that the simulated code is only a model of the actual code that executes in the sensor nodes and in the robots. However, since C is the programming language used in both cases the translation is in most cases quite straightforward.

In spite of the above it is our experience that the TrueTime simulation approach gives results that are close to the real case. The TrueTime approach has also been validated by others. In [11] a TrueTime-based model is compared with a hardware-in-the-loop (HIL) model of a distributed CAN-based control system. The TrueTime simulation result matched the HIL results very well.

An aspect of the model that is extremely difficult, if not impossible, to validate is the wireless communication. Simulation of wireless MANET systems is notoriously difficult, see, e.g., [12]. The effects of multi-path propagation, fading, and external disturbances are very difficult to model accurately. The approach adopted here is to first start with an idealized exponential decay radio model and then when

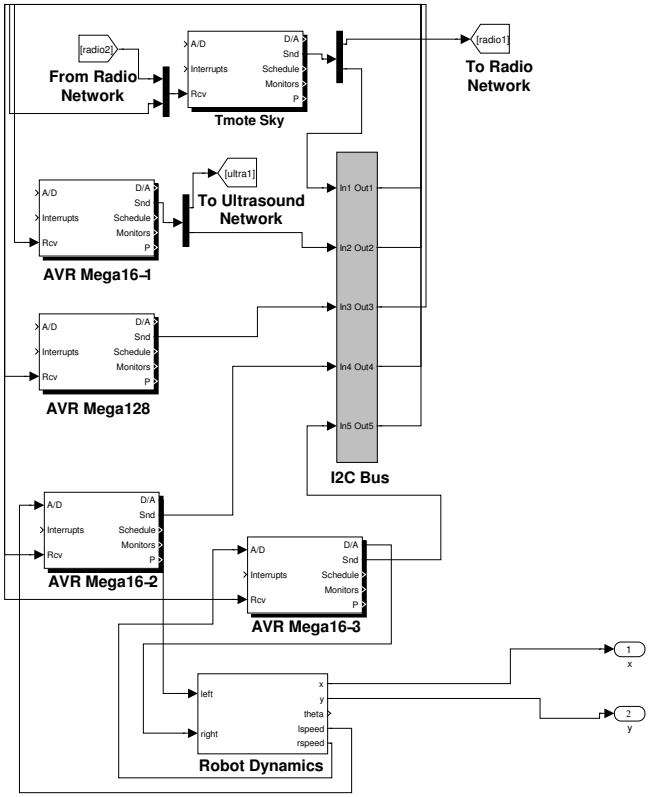

Fig. 7. The Simulink model of the mobile robots. For the sake of clarity the obstacle detection sensors have been omitted. These should be connected to AVR Mega16-1.

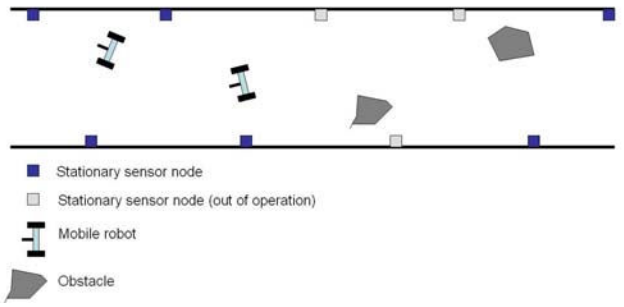

Fig. 8. Animation workspace

this works properly gradually add more and more non-determinism. This is done either by setting a high probability that a packet is lost, or by providing a user-defined radio model using Rayleigh fading.

X. Related Work

There exists a large number of general network simulators today. One of the most well-known is ns-2 [13], which is a discrete-event simulator for both wired and wireless networks with support for, e.g., TCP, UDP, routing, and multicast protocols. It also supports simple movement models for mobile applications, where the position and velocity of nodes may be specified in a script. It should be noted that the default radio model in ns-2 is very simplistic (even more simplistic than TrueTime's), although more accurate physical layer models may be implemented [14].

Another discrete-event computer network simulator is OMNeT++ [15]. It contains detailed IP, TCP, and FDDI protocol models and several other simulation models (file system simulator, Ethernet, framework for simulation of mobility, etc.). Compared to these simulators, the network simulation part in Truetime is far more simplistic. However, the strength of TrueTime is the co-simulation facilities that makes it possible to simulate the latency-related aspects of the network communication in combination with the node computations and the dynamics of the physical environment. Rather than basing the co-simulation tool on a general network simulator and then try to extend this with additional co-simulation facilities, the approach has been to base the co-simulation tool on a powerful simulator for general dynamical systems, i.e., Simulink, and then add support for simulation of real-time kernels and the latency aspects of network communication to this. An additional advantage of this approach is the possibility to make use of the wide range of toolboxes that are available for Matlab/Simulink. For example, support for virtual reality animation.

There are also some network simulators geared towards the sensor network domain. TOSSIM [16] compiles directly from TinyOS code and scales very well. The COOJA simulator [17] makes it possible to simulate sensor networks running the Contiki OS. Network in a box (NAB) [18] is another simulator for large-scale sensor networks. Another example is J-Sim, a general compositional simulation environment that includes a generalized packet switched network model that may be used to simulate wireless LANs and sensor network [19]. Again, these types of simulators generally lack the possibility to simulate continuous-time dynamics, that is present in TrueTime.

Another type of related tools are software emulators such as, e.g., the Simics system [20]. Although systems of this type provide very accurate ways of simulating software they, generally, have weak support for networks and continuous-time dynamics.

A few other tools have been developed that support co-simulation of real-time computing systems and control systems. RTSIM [21] has a module that allows system dynamics to be simulated in parallel with scheduling algorithms. XILO [22] supports the simulation of system dynamics, CAN networks, and priority-preemptive scheduling. Ptolemy II is a general purpose multi-domain modeling and simulation environment that includes a continuous-time domain, and a simple RTOS domain. Recently it has been extended in the sensor network direction [23]. In [24] a co-simulation environment based on ns-2 is presented. The ns-2 simulator has been extended with an ODE-solver for dynamical simulations of the controller units and the environment. However, this tool lacks support for real-time kernel simulation.

XI. Conclusions

New simulation tools such as TrueTime are needed to capture the complex interactions that exist between hardware,

software, and the physical environment in wireless sensor/actuator applications. The TrueTime approach is based on co-simulation of (sometimes simplistic) models of the environment, the code executing inside the nodes, and the network communication. In this paper it has been described how TrueTime can be used to model and simulate several aspectes of a large road tunnel disaster relief scenario involving mobile robots. Since TrueTime is based on MATLAB/Simulink, the modeling of the robot dynamics and the environment is straightforward. The TMote Sky nodes and ATMEL AVR microcontrollers can be modeled by the TrueTime kernel blocks, while I^2C buses and the Zigbee radio communication is modeled by TrueTime wired and wireless network blocks. Finally, a new TrueTime ultrasound block has been developed to faciliate the simulation of the RUNES localization component.

A. Acknowledgment

The work has been done with partial support from the EC project RUNES (Contract IST-2004-004536).

REFERENCES

[1] K.-E. Årzén, A. Bicchi, G. Dini, S. Hailes, K. Johansson, J. Lygeros, and A. Tzes, "A component-based approach to the design of networked control systems," in *Proceedings of the European Control Conference (ECC), Kos, Greece*, Kos, Greece, 2007.

[2] 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 *Proceedings of the European Control Conference (ECC), Kos, Greece*, Kos, Greece, 2007.

[3] A. Panousopoulou and A. Tzes, "Utilization of mobile agents for Voronoi-based heterogeneous wireless sensor network reconfiguration," in *Proceedings of the European Control Conference (ECC), Kos, Greece*, Kos, Greece, 2007.

[4] B. Zurita Ares, C. Fischione, A. Speranzon, and K. Johansson, "On power control for wireless sensor networks: radio model, software implementation and experimental evaluation," in *Proceedings of the European Control Conference (ECC), Kos, Greece*, Kos, Greece, 2007.

[5] A. Cervin, D. Henriksson, B. Lincoln, J. Eker, and K.-E. Årzén, "How does control timing affect performance?" *IEEE Control Systems Magazine*, vol. 23, no. 3, pp. 16–30, June 2003.

[6] 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, Dec. 2005.

[7] A. Cervin, J. Eker, B. Bernhardsson, and K.-E. Årzén, "Feedback-feedforward scheduling of control tasks," *Real-Time Systems*, vol. 23, no. 1–2, pp. 25–53, July 2002.

[8] A. Dunkels, B. Grönvall, and T. Voigt, "Contiki - a lightweight and flexible operating system for tiny networked sensors," in *Proceedings of the First IEEE Workshop on Embedded Networked Sensors (Emnets-I)*, Tampa, Florida, USA, Nov. 2004. [Online]. Available: http://www.sics.se/~adam/dunkels04contiki.pdf

[9] A. Dunkels, O. Schmidt, T. Voigt, and M. Ali, "Protothreads: Simplifying event-driven programming of memory-constrained embedded systems," in *Proceedings of the Fourth ACM Conference on Embedded Networked Sensor Systems (SenSys 2006)*, Boulder, Colorado, USA, Nov. 2006. [Online]. Available: http://www.sics.se/~adam/dunkels06protothreads.pdf

[10] C. Perkins and E. 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.

[11] D. Ayavoo, M. Pont, and S. Parker, "Using simulation to support the design of distributed embedded control systems: a case study," in *In Proceedings of 1st UK Embedded Forum*, Brimingham, UK, 2004.

[12] T. Andel and A. Yasinac, "On the credibility of manet simulations," *IEEE Computer*, pp. 48–54, July 2006.

[13] "Ns-2," Home page: http://www.isi.edu/nsnam/ns/index.html, 2004.

[14] J.-M. Dricot and P. De Doncker, "High-accuracy physical layer model for wireless network simulations in NS-2," in *In Proceedings of the Int. Workshop on Wireless Ad-Hoc Networks (IWWAN)*, Oulu, Finland, 2004.

[15] "Omnet++," Home page: http://www.omnetpp.org, 2004.

[16] P. Levis, N. Lee, M. Welsh, and D. Culler, "TOSSIM: accurate and scalable simulation of entire TinyOS applications," in *Proceedings of the 1st international conference on Embedded networked sensor systems*, Los Angeles, CA, USA, 2003, pp. 126–137.

[17] F. Österlind, "A Sensor Network Simulator for the Contiki OS," SICS – Swedish Institute of Computer Science, Tech. Rep. T2006-05, Feb. 2006. [Online]. Available: ftp://ftp.sics.se/pub/SICS-reports/Reports/SICS-T--2006-05--SE.pdf

[18] "NAB (Network in A Box)," Home page: http://nab.epfl.ch/, 2004.

[19] H.-Y. Tyan, "Design, realization and evaluation of a component-based compositional software architecture for network simulation," Ph.D. dissertation, Ohio State University, 2002.

[20] P. Magnusson, "Simulation of parallel hardware," in *In Proceedings of the Int. Workshop on Modeling Analysis and Simulation of Computer And Telecommunication Systems (MASCOTS)*, San Diego, CA, 1993.

[21] L. Palopoli, L. Abeni, and G. Buttazzo, "Real-time control system analysis: An integrated approach," in *Proceedings of the 21st IEEE Real-Time Systems Symposium*, Orlando, Florida, December 2000.

[22] J. El-Khoury and M. Törngren, "Towards a toolset for architectural design of distributed real-time control systems," in *Proceedings of the 22nd IEEE Real-Time Systems Symposium*, London, England, December 2001.

[23] P. Baldwin, S. Kohli, E. A. Lee, X. Liu, and Y. Zhao, "Modeling of sensor nets in Ptolemy II," in *IPSN'04: Proceedings of the third international symposium on Information processing in sensor networks*. ACM Press, 2004, pp. 359–368.

[24] M. Branicky, V. Liberatore, and S. M. Phillips, "Networked control systems co-simulation for co-design," in *Proc. American Control Conference*, 2003.