Holistic Simulation of Mobile Robot and Sensor Network Applications Using TrueTime

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

2007

Link to publication

Citation for published version (APA):

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

• Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
• You may not further distribute the material or use it for any profit-making activity or commercial gain
• You may freely distribute the URL identifying the publication in the public portal

Take down policy
If you believe that this document breaches copyright please contact us providing details, and we will remove access to the work immediately and investigate your claim.
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 applica-
tion areas for embedded real-time systems where wire-
less 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 environ-
ment. One example is a sensor network that monitors the
presence of moving objects in the environment. Other exam-
ple 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 (Re-
configurable 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 wire-
less 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 micro-
controllers 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 sim-
ulation 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 communica-
tion and the node computations. Hence, something more is
needed. TrueTime [5], [6] is a MATLAB/Simulink-based co-
simulation 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 distur-
bance 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
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$^2$C 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.

The RBbots internally consists of one Tmote Sky, one ATME AVR Mega128, and three ATME AVR Mega16 microprocessors. The nodes communicate internally over an I$^2$C bus. The Tmote Sky is used for the radio communication as the master. Two of the ATME AVR Mega16 processors are used as interfaces to the wheel motors and the wheel encoders measuring the wheel angular velocities. The third ATME 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 ATME 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,
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.

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.

Fig. 4. RBbot hardware architecture.
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 I²C 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 I²C 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 I²C 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 I²C 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 I²C bus share connection pins. Due to this it is only possible to have one master on the I²C 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 I²C bus is modeled as a CAN bus with the transmission rate set to match the transmission rate of the I²C 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 packet 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 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.
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 corresponding 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

\[
\begin{align*}
\dot{x} &= \frac{1}{2}(R_1\omega_1 + R_2\omega_2)\cos(\theta) \\
\dot{y} &= \frac{1}{2}(R_1\omega_1 + R_2\omega_2)\sin(\theta) \\
\dot{\theta} &= \frac{1}{D}(R_2\omega_2 - R_1\omega_1)
\end{align*}
\]

(1)

where the state consists of the \(x\)- and \(y\)-positions and the heading \(\theta\). Input to the system are the angular velocities \(\omega_1\) and \(\omega_2\) 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
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 aspects of a large road tunnel disaster relief scenario involving mobile robots. Since TrueTime is based on MAT-LAB/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 1C 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 facilitate 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


