A Simulation-Based Virtual Environment to Study Cooperative Robotic Systems

Xiaolin Hu (corresponding author)
Computer Science Department, Georgia State University, Atlanta GA, USA 30303

Address: Department of Computer Science, Georgia State University, 34 Peachtree Street, Suite 1450, Atlanta, GA30303
Phone: 404-463-9857
Fax: 404-463-9912
Email: xhu@cs.gsu.edu

Bernard P. Zeigler
Arizona Center for Integrative Modeling and Simulation, University of Arizona, Tucson AZ, USA 85721

Address: Electrical & Computer Engineering Dept., University of Arizona, 1230 E. Speedway Blvd, Tucson, AZ 85721-0104
Phone: (520) 626-4846
Fax: (520) 621-6184
Email: zeigler@ece.arizona.edu

ABSTRACT

Simulation plays important roles in experimenting with, understanding, and evaluating the performance of cooperative robotic systems. Typically, simulation-based studied of robotic systems are conducted in the computer, without involving any real system components. This paper presents a new hybrid approach to simulation that allows real robots as well as robot models to work together in a simulation-based virtual environment. This capability of robot-in-the-loop simulation effectively bridges conventional simulation and real system execution, augmenting them to constitute an incremental study and measurement process. It is especially useful for large-scale cooperative robotic systems whose complexity and scalability severely limit experimentation in a physical environment using real robots. We present architecture for this simulation-based virtual environment that, together with an incremental study process and associated experimental frames, can support systematic analysis of cooperative robotic systems. An example of robotic convoy is provided. Some measurement metrics are developed and simulation results are described. An experimental setup for robot-in-the-loop simulation is discussed.

KEYWORDS: *Virtual Environment, Robot-in-the-Loop Simulation, Incremental Study Process, Experimental Frame, Model Continuity, DEVS, Cooperative Robotic System, Robotic Convoy*

1. INTRODUCTION

Cooperative robotic systems couple computational intelligence to the physical world. These systems consist of multiple homogenous or heterogeneous robots that perceive the environment, make decisions, and carry out commands to affect the environment. Communication and cooperation is important for theses systems since robots work as a collective team to accomplish common tasks. Several taxonomies and metrics have been defined to classify such systems. For example, Dudek, *etc.* [10] classify robotic collectives along seven dimensions: size of the collective, communication range, communication topology, communication bandwidth, collective reconfigurability, processing ability of each collective unit, and collective composition. Balch [1] classifies the performance metric of multirobot tasks based on time, subject of action, resource limits, group movement, platform capabilities, *etc*.

The increasing complexity of cooperative robotic systems calls for systematic methods as well as supporting environments to experiment with, understand, and evaluate them. To serve this purpose, modeling and simulation technologies are frequently applied. With simulation-based methods, robot models including sensor and actuator models are developed and then simulated on computers. Different experimental configurations can be easily applied to test and evaluate the system under development. To support simulation of a robotic system that actively interacts with an external environment, an environment model is usually developed. This environment model serves as a "virtual" environment to provide sensory input to robot models and to responr to robots' actuations. For example, it may include virtual obstacles that can be sensed when robot models move around. In this conventional way of simulation, all involved components are models and the simulation-based study is conducted in a modeled world at a certain abstraction level.

To bring simulation-based study closer to the reality, we developed a hybrid simulation approach that allows real robots to be included in a simulation by replacing some of the robot models with real counterparts. "Robot-in-the-loop" simulation refers to this capability of experimenting with several real robots in a simulation-based virtual environment containing many robot models. This capability, which effectively bridges conventional simulation and real system execution, is especially useful for large-scale cooperative robotic systems whose complexity and scalability severely limit experimentations in a physical environment using real robots. This research is an extension to our previous work on a "model continuity" software development methodology for distributed real time systems [14, 16]. It is based on the DEVS (Discrete Event System Specification) [28] modeling and simulation framework.

The rest of this paper is organized as follows. Section 2 introduces the related work and background of this research. They include related work on robot simulation and virtual environments and background on model continuity methodology and DEVS. Section 3 presents the simulation-based virtual environment from three aspects: the architecture, an incremental measuring process, and experimental frames. Section 4 describes a robotic convoy system as an illustrative example. The system model is described; several performance metrics and their simulation results are presented; and an experimental setup of "robot-in-the-loop" simulation is discussed. Section 5 concludes this work and discusses some future research directions.

2. Related Work and Background

Many multi-robot simulators have been developed during the last 10 years. For example, MuRoS [5] is an object-oriented simulator developed at University of Pennsylvania. It supports simulation of several multi-robot applications such as cooperative manipulation, formation control, foraging, etc. The Stage [25] is a robot simulator that simulates a population of mobile robots, sensors and objects in a two-dimensional bitmapped environment. It is designed to support research into multi-agent autonomous systems, so it provides fairly simple, computationally cheap models of lots of devices rather than attempting to emulate any device with great fidelity. Unlike the 2D multi-robot simulators mentioned above, ÜberSim [3] provides 3D simulation and is specifically targeted for simulating games of robot soccer. ÜberSim emphasizes capturing the effects of dynamics, friction, and collisions that exist in the environment by using a physics simulation engine. Other 3D multiple robot simulators include Gazebo [13], which is a multi-robot simulator for outdoor environments. Like the Stage mentioned above, Gazebo is capable of simulating a population of robots, sensors and objects, but does so in a three-dimensional world. It generates both realistic sensor feedback and physically plausible interactions between objects (it includes an accurate simulation of rigid-body physics). A commercial 3D simulating package worth of mentioning is Webots [18], which is a simulator originally developed for the Kephera robot but now able to support other types of mobile robots including wheeled and legged robots. The Webots simulation software provides users with a rapid prototyping environment for modeling, programming and simulating mobile robots. None of these multi-robot simulations, however, involves any real system components such as real sensors, real actuators, or real robots. Our research, by taking a hybrid approach, allows real robots as well as robot models to be studied together in a simulation-based virtual environment. It also emphasizes the continuous transition from simulation-based study to real robot execution.

The concept of virtual environment is usually associated with the technology of virtual reality (VR) [20, 4], which has been applied to various areas such as simulation of manufacturing plants, the planning of robotic workcells, and robot teleoperation systems [23, 11]. While the research on VR mainly deals with the interaction with human operators, our work focuses on the interaction between robots and the virtual environment. The following research work is related to our research from this perspective. Komoriya and Tani [17] developed a system that allows a single real robot to be tested and experimented in a virtual environment. This serves the purpose to avoid any damages that may happen during the process of testing and refining the control logic of the robot. Wang [26] proposed a simulation environment that allows a real robot to be integrated into distributed robotic system simulation, thus making the simulation more realistic. The work of RAVE [9] developed a simulation environment that supports simulated robots to interact with real robots and the ability to place virtual sensors on real robots to augment or experiment with their performance. Our research extends these works by allowing one or multiple real robots to interact with a virtual environment and to communicate and cooperate with robot models. A well-defined architecture and an incremental study process have been developed to support the study of cooperative robotic systems in a systematic way.

This research is an extension to our previous work on a "model continuity" software development methodology for distributed real-time systems [14, 16]. In general, model continuity refers to the ability to transition as much as possible of a model specification through the stages of a development process. Our work on model continuity has focused on the transition from designed models to the final software in a systematic way. Model continuity allows the decision making models of a distributed real-time system to be tested incrementally, and then deployed to a distribute environment for execution. It supports a design and test process having 4 steps: 1) Centralized simulation to analyze the system under test within a model of the environment linked by abstract sensor/actuator interfaces on a single computer. 2) Distributed simulation, in which models are deployed to the real network that the system will be executed on and simulated by distributed real-time simulators. 3) Hardware-in-the-loop (HIL) simulation, in which the environment model is simulated by a DEVS real-time simulator on one computer while the control model under test is executed by a DEVS real-time execution engine on the real hardware. 4) Real execution, in which a real-time execution engine executes the control model that interacts with the real environment through the earlier sensor/actuator interfaces that have been appropriately instantiated. Model continuity limits the possibility of design discrepancy along the development process, thus increases the confidence that the final system realizes the specification as desired. Furthermore, it makes the design process easier to manage since continuity between models of different design stages is retained. This methodology, when applied to distributed robotic system, greatly eases the transition from simulation-based study to real robot implementation [15].

The "model continuity" methodology and the research presented in this paper are based on the DEVS (Discrete Event System Specification) modeling and simulation framework [28]. The DEVS formalism is derived from generic dynamic systems theory and has been applied to both continuous and discrete phenomena. It provides a formal modeling and simulation (M&S) framework with well-defined concepts of coupling of components, and hierarchical modular model construction. DEVS has been operationalized to serve as a practical simulation and execution tool in a variety of implementations [8, 7, 27].

DEVS M&S supports two kinds of DEVS models: *atomic model* and *coupled model*. Atomic models are the basic components. Coupled models have multiple sub-components and can be constructed in a hierarchical way. An atomic model is defined as:

$$M = <X, S, Y, \delta_{int}, \delta_{ext}, \lambda, ta>$$

where

X is the set of input values.

S is a set of states.

Y is the set of output values.

$\delta_{int}: S \rightarrow S$ is the *internal transition* function.

$\delta_{ext}: Q \times X^b \rightarrow S$ is the *external transition* function, where

$Q \in \{(s,e) \mid s \in S, 0 \leq e \leq ta(s)\}$ is the *total state* set,

e is the *time elapsed since* last transition,

X^b denotes the *collection* of bags over X (sets in which some elements may occur more than once).

$\lambda: S \rightarrow Y^b$ is the output function.

ta: $S \rightarrow \mathrm{R}^{+}_{0,\infty}$ is the *time advance* function.

The interpretation of these elements is briefly described below. The input event set defines all possible inputs that may occur on the input ports; the output set consists of all possible outputs the atomic model may send out. External inputs received on the input ports invoke a model's external transition function, which determines how the model changes its state based on the inputs and model's current state. The model remains in a state for an amount of time that is determined by the time advance function. When this time has expired the output function is invoked, which sends output on the output ports. Following the invocation of the output function, the internal transition function is invoked to transit the model to a new state.

Atomic models may be coupled in the DEVS formalism to form a coupled model. A coupled model tells how to couple (connect) several component models together to form a new model. This latter model can itself be employed as a component in a larger coupled model, thus giving rise to hierarchical construction. A coupled model contains the following information: the set of *components*; the set of *input ports*; the set of *output ports*; and the *coupling specification* that defines the connections between models.

An atomic model may start an *Activity,* which is a concept initially introduced by RT-DEVS for real-time systems specification [12]. A DEVS activity can be any kind of computation task. It belongs to an atomic model and may return computational results to the atomic model. If an activity returns results to an atomic model, the result will be put on a reserved input port (the "*outputFromActivity*" port) as an external event and then be processed by the model's external transition function. In this research, activities are used to model or represent robots' sensors and actuators. More information about different kinds of sensor/actuator activities can be found in section 3.1.2.

3. A VIRTUAL ENVIRONMENT FOR ROBOTIC SYSTEMS

The effectiveness of this simulation-based virtual environment is supported by a well-defined architecture, an incremental study process, and by integrating experimental frames to specify metrics for performance measurement. Next we present these three aspects respectively.

3.1 Architecture of the Simulation-based Virtual Environment

Robotic systems can be viewed as a particular form of real-time systems that monitor, respond to, or control, an external environment. This environment is connected to the computer system through sensors, actuators, and other input-output interfaces [24]. A robotic system from this point of view consists of sensors, actuators and the decision-making unit. A cooperative robotic system is composed of a collection of robots that communicate with each other and interact with an environment.

This view of robotic systems suggests a basic architecture for the simulation-based virtual environment that we developed: an environment model and a collection of robot models, which include decision-making models, sensors, and actuators. The environment model represents the real environment within which a robotic system will be executed. It forms a virtual environment for the robots and may include virtual obstacles, virtual robots, or any other entities that are useful for simulation-based study. The robot model represents the control software that governs a robot's decision-making and communication. It also includes sensor and actuator interfaces to support interactions between the robot and its environment. By following the "model continuity" methodology, we clearly separate a robot's decision-making unit, which is modeled as a DEVS atomic or coupled model, from the sensors and actuators that are modeled as DEVS *abstractActivities*. The decision-making model defines the control logic. The sensor/actuator *abstractActivities* represent the sensors or actuators, including their behaviors, interfaces, and properties of uncertainty and inaccuracy. Couplings are added between sensor/actuator *abstractActivities* and the environment model, so messages can be passed between the decision-making model and the environment model through sensor/actuator *abstractActivities*.

3.1.1 Robot-in-the-loop simulation

The clear separation between robots' decision-making model and sensor/actuator interfaces brings several advantages. First, it separates a robot's decision-making from hardware interaction. This makes it easier for the designer to focus on the decision-making model, which is the main design interest. Secondly, the existence of a sensor/actuator interface layer makes it possible for the decision-making model to interact with different types of sensors/actuators, as long as the interface functions between them are maintained the same. Thus depending on different experimental and study objectives, a decision-making model can be equipped with different types of sensors and actuators. In our research, models of sensors/actuators (modeled as *abstractActivities*, also referred as virtual sensors/actuators hereafter) are developed to simulate the behavior of real sensors/actuators.

Meanwhile, real sensor/actuator interfaces (implemented as *RTActivities*) are developed to drive the real sensor/actuators of a robot. A virtual sensor/actuator and its corresponding real sensor/actuator interface share the same interface functions with the decision-making model. During simulation, a decision-making model uses virtual sensors/actuators to interact with a virtual environment; during real execution, the same decision-making model, which resides on a real robot, uses real sensor/actuator interfaces to interact with a real environment.

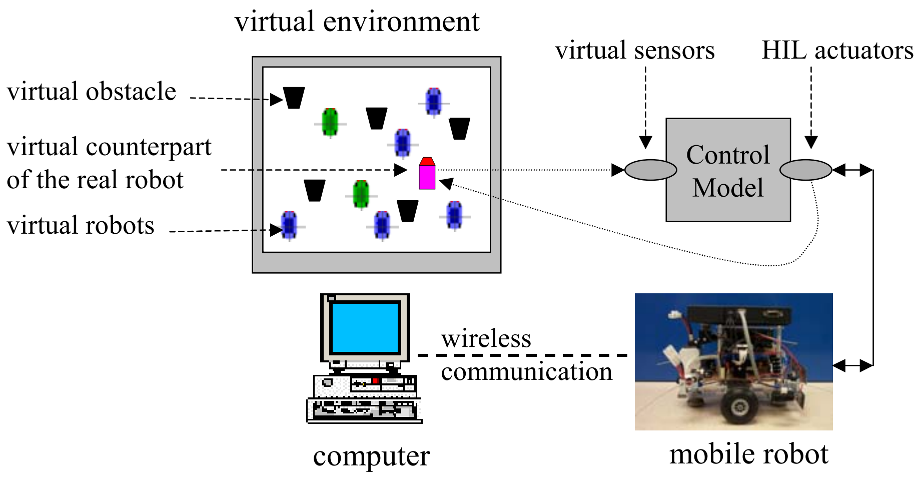


Figure 1: Architecture of robot-in-the-loop simulation

An intermediate stage is also developed to allow the decision-making model on a real robot to interact with a virtual environment that is simulated on a computer. We call this stage robot-in-the-loop simulation. It is achieved by configuring a real robot to use a combination of virtual and real sensors/actuators. For example, Figure 1 shows such a setup where one real mobile robot works with a virtual environment. In this example, the mobile robot uses its virtual sensors to get sensory input from the virtual environment and uses its real motor interface to move the robot. As a result, this real robot moves in a physical space based on the sensory input from a virtual environment. Within this virtual environment, the robot "sees" virtual obstacles that are simulated by computers and makes decisions based on those inputs. Meanwhile, virtual robots (robot models) can also be added into the environment so the real robot can sense them and communicate/coordinate with them. This capability of robot-in-the-loop simulation brings simulation-based study one-step closer to the reality. Furthermore, for large-scale cooperative robotic systems that include hundreds of robots, it makes it possible to conduct system-wide tests and measurements without waiting for all real robots to be available. In this later case, the rest of robots can be provided by the simulation-based virtual environment.

3.1.2 Synchronization between real robots and the virtual environment

Including real robots into simulation-based study brings an important issue that needs to be resolved: the synchronization between real robots and the virtual environment. For example, in Figure 1, when the decision-making model issues a moving command, the real robot will move a distance in the physical space. This position change needs to be updated by the virtual environment too. For this purpose, each real robot has a virtual counterpart in the virtual environment. When a real robot moves, the position of its virtual counterpart will be updated. Thus synchronization between the real robot and the virtual environment is actually the synchronization between the real robot and its virtual counterpart.

To synchronize a real robot and its virtual counterpart, HIL (hardware-in-the-loop) sensors/actuator Activities (*HILActivities*) have been developed. Different from a sensor/actuator *abstractActivity* that is coupled to the virtual environment and different from a real sensor/actuator interface *RTActivity* that interacts with a real environment, a sensors/actuator *HILActivity* is a combination of both. It drives the real sensor/actuator to interact with a real environment, while in the meantime is coupled to the environment model. This allows the real robot to send messages to the environment model to achieve synchronization. For the example shown in Figure 1, a motor *HILActivity* drives the robot to move a distance. When the desired distance is reached[1], a *moveComplete* message will be triggered by the motor and will be captured by the motor *HILActivity*. The motor *HILActivity* then sends a message to the environment model, which updates the position of the virtual counterpart of the real robot based on the received moving parameter. We note that this approach does not count the error of the robot's motion, thus as time proceeds this error will accumulate. A more advanced synchronization mechanism may be developed by using a separated monitoring system to track the motion of real robots. It then updates the information, such as the distance and time of a robot's movement, to the environment model to synchronize them. This is the approach taken by [17] where an overhead camera is used to record robot's movement and then synchronize with the environment model.

[1] In this example, wheel encoders are used to determine if desired distance is reached.

As described above, three types of DEVS activities have been developed to act as sensors/actuators interfaces between the decision-making model and the environment model. They are abstract Activity *abstractActivity*, real-time Activity *RTActivity*, and hardware-In-the-Loop Activity *HILActivity*. These activities play different roles in different conditions. An *abstractActivity* serves as a virtual sensor or actuator that is used by the decision-making model to interact with the environment model in simulation. An *RTActivity* is used in real execution to drive a real sensor or actuator. A *HILActivity* is employed in robot-in-the-loop simulation to drive a real sensor or actuator and also synchronizes with the environment model. It is important to note that in order to maintain the decision making model unchanged in different conditions, the corresponding *abstractActivity*, *RTActivity*, and *HILActivity* should maintain a same set of interface functions with the decision-making model.

Another synchronization issue is the time synchronization problem between virtual environment's simulation and real robots' execution. For the same reason that a flight simulator that trains pilots need to run in real time, the simulation of the virtual environment that interact with real robots needs to run in real time too. This is especially critical when a real robot makes decisions in a timely fashion. For example, a robot may monitor the change of a sensory input within a time window, saying 10 seconds, to make a decision. In order for the robot to make a "correct" decision, the simulation of the virtual environment should run at the same speed of the world clock time. Our previous work on real-time simulation and distributed simulation has developed several techniques to address this time synchronization problem between simulation and the real world [6, 7].

3.2 From Robot Model To Real Robot – An Incremental Study Process

By supporting combined robot models and real robots to work together, this virtual environment adds to the conventional simulation-based study to form an incremental study process. This study process includes three steps: conventional simulation, robot-in-the loop simulation, and real robot execution. Figure 2 illustrates this process by considering a system with two robots.

The first step is conventional simulation, where all components are models that are simulated by fast-mode or real-time simulators on one computer. As shown in Figure 2(a), in this step both robot models are equipped with virtual sensors and virtual actuators (*abstractActivities*) to interact with a virtual environment (environment model). Couplings between two robots can be added so they can communicate to each other. This conventional way of simulation has the most flexibility because all components are models so different configurations can be easily applied to study the system under development.

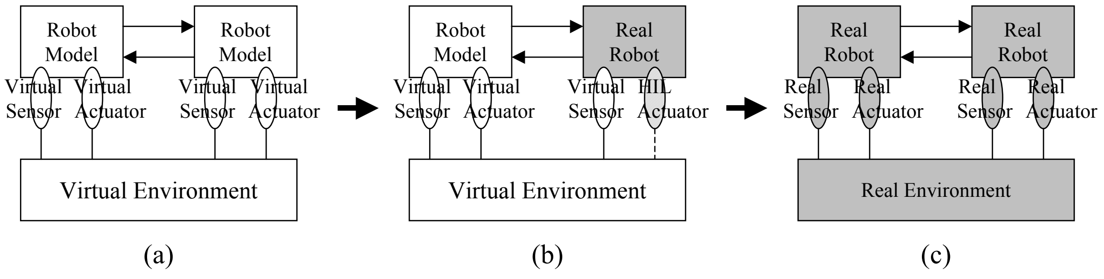


Figure 2: An incremental study process

The second step is robot-in-the-loop simulation where one or more real robots are studied within a virtual environment together with other robot models that are simulated on computers. By replacing robot models with real robots, this step brings simulation-based study closer to the reality and increases the fidelity of simulation results[2]. As shown in Figure 2(b), in this step the robot model still use virtual sensors/actuators. However, depending on the study objectives, the real robots may have a combination of virtual sensors/actuators and HIL sensors/actuators (*HILActivities*). For example, the real robot shown in Figure 2(b) uses a virtual sensor and a HIL actuator. Through virtual sensor, it gets sensory input from the virtual environment. Using HIL actuator, it interacts with a real environment (the real environment is not shown in the figure) and also synchronizes with the virtual environment. In robot-in-the-loop simulation, robot models and the virtual environment are simulated on a computer; the model that controls the real robots runs on the real robot. The couplings between the two robots are maintained the same, so the real and virtual robots can interact with each other in the same way as that in the first step. However here the real commutation actually happens across a network. As mentioned before, since real robots are involved in the simulation, robot-in-the-loop simulation needs to run in a real-time fashion.

The final step is the real system study, where all real robots run in a real environment. These robots use real sensors and actuators (*RTActivities*). They communicate in the same way as that from the first two steps because the couplings between them are not changed through the process. Since all measurement results of this step come

[2] This may not hold true if robot models have considered enough details of real robots.

directly from the real system, they have the most fidelity. However, they are also most costly and time consuming to be collected.

This incremental simulation-based study process establishes an operational procedure to measure and evaluate cooperative robotic systems. As the process proceeds, the flexibility (easy to setup different experiments) and productivity (time saving and cost saving) of the measurement decreases and the fidelity (loyal to the reality) of the measurement increases.

3.3. Specify Measurement Metrics Using Experimental Frame

Metrics and analysis play important roles in simulation-based study. To support systematic analysis of a robotic system under development, the simulation-based virtual environment includes experimental frames [28] as important components. An experimental frame (EF) is a specification of the conditions within which the system is observed or experimented. In DEVS-based modeling and simulation framework, an experimental frame is realized as a system that interacts with the source system, or System Under Test (SUT), to obtain the data of interest under specified conditions. It consists of four major subsections:

- *input stimuli*: specification of the class of admissible input time-dependent stimuli. This is the class from which individual samples will be drawn and injected into the model or system under test for particular experiments.
- *control*: specification of the conditions under which the model or system will be initialized, continued under examination, and terminated.
- *metrics*: specification of the data summarization functions and the measures to be employed to provide quantitative or qualitative measures of the input/output behavior of the model. Examples of such metrics are performance indices, goodness-of-fit criteria, and error accuracy bound.
- *analysis*: specification of means by which the results of data collection in the frame will be analyzed to arrive at final conclusions. The data collected in a frame consists of pairs of input/output time functions.

When an experimental frame is realized as a system to interact with the SUT (or its model), the four specifications become components of the driving system. For example, a generator of output time functions implements the class of input stimuli.

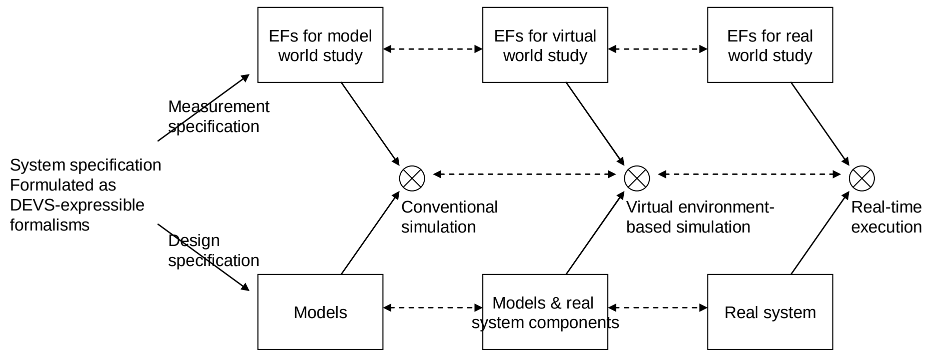

EFs for model world study
EFs for virtual world study
EFs for real world study
Measurement specification
System specification Formulated as DEVS-expressible formalisms
Design specification
Conventional simulation
Virtual environment-based simulation
Real-time execution
Models
Models & real system components
Real system

Figure 3: Experimental frames (EF), models, and study methods

Integrating experimental frames into the incremental study process described in section 3.2 brings the advantage that measurement metrics can be formally specified. Since three stages exist in this process, EFs can be developed corresponding to the three stages. Following this idea, Figure 3 illustrates how experimental frames, models/systems, and simulation methods can play together to carry out simulation-based measurement. This process includes three lines of development and integration: the *models/system* that will be tested and measured; the *experimental frames* that specify the measurement; and the *methods* that are employed to carry out the measurement. The process starts from the system specification that is formulated as DEVS expressible formalisms. The system specification is further divided into two specifications: the design specification that guides the development of models/system, and the measurement specification that guides the development of experimental frames. Three methods, corresponding to three stages of study, are used to carry out the measurement incrementally. These methods are conventional simulation, virtual environment-based simulation, and real time execution. Similarly, three types of experimental frames exist: EF for model world study, EF for virtual world study, and EF for real world study. Techniques are under development to derive experimental frame development from the measurement specification in automated or semi-automated ways.

To develop experimental frames for large-scale robotic systems, several more steps can be added to match the system's development evolution starting from a single robot, to small team, to large team, to combined real and virtual robots, and to the final system with all real robots. The strategy is to categorize experimental frames in several groups that correspond to the levels of measurement. For example, for a cooperative robotic system that include a large number (saying 1000) of homogeneous mobile robots, the following experimental frames can be developed.

1. EF for *developmental* testing of single robot – focus on achieving correct behavioral logic of control models within a single robot; derive EF tests from logical specifications in semi-automated manner. Tests are intended to provide coverage of all input/output sequences apt to occur when the robot is embedded in real environment.
2. EF for *developmental* testing of small robotic team – focus on achieving correct interplay among small robotic team compositions (e.g., sizes 2-5); derive EF tests from robot interaction protocols in semi-automated manner. In addition, component-level measures of performance are defined and collected by the EF in observing a robotic team.
3. EF for *operational* testing of large robotic team – focus is on both behavior and performance of moderate to full-scale robotic team compositions (e.g., sizes 6-1000). EFs are based on scenarios that are expected to occur in real applications. Emphasis shifts from I/O testing to collecting holistic measurement data that can be employed for diagnosis of problems and to assess overall task effectiveness.
4. EF for *operational* testing of large robotic team with hybrid real and virtual robots – focus on achieving correct behavior of real robots and verifying system's performance in a hybrid environment. EFs are based on real robot scenarios as well as the EFs from the operational testing in the previous step.
5. EF for *operational* testing of final system– Full-scale system in real environment reduces level of experimental control to infrequent interventions to steer system toward desired modes. Concomitantly, emphasis shifts further to collecting significant and detailed measurement data to enable evaluation of overall system effectiveness in after action review.

4. ROBOT CONVOY: A CASE STUDY EXAMPLE

This simulation-based virtual environment has supported the development of a scalable robot convoy system. Below we describe the model of this system, some measurement metrics and simulation results, and an experimental setup of robot-in-the-loop simulation for this system.

4.1 System Description and Model

This robot convoy system consists of an indefinite number of robots, say *N* robots (*N*>1). These robots are in a line formation where each robot (except the *leader* and the *ender*) has a front neighbor and a back neighbor. The robots used in this system are car type mobile robots with wireless communication capability. They can move forward/backward and rotate around the center, and have whisker sensors and infrared sensors [21, 22].

One of the main goals of this convoy system is to maintain the coherence of the line formation and to synchronize robots' movements. Synchronization means a robot cannot move forward if its "front" robot doesn't move, and it has to wait if its "back" robot doesn't catch up. To serve this purpose, synchronization messages are passed between a robot and its neighbors. To achieve coherence of the line formation, the moving parameters of a "front" robot are passed back to its immediate back robot. This allows the back robot to plan its movement accordingly based on its front robot's movement. The system has no global communication and coordination as one objective is to study how global behavior may be achieved by using localized sensing and communication.

System Model

Figure 4 shows the model of this system. Each block represents a robot model, which is a DEVS coupled model. The system model includes *N* sub-models that are corresponding to the *N* robots in the system. Each intermediate robot model has two input ports: *FReadyIn*, *BReadyIn* and two output ports: *FReadyOut*, *BReadyOut*. These ports are used to send/receive synchronization messages between robots and to pass moving parameters from a "front" robot to the "back" robot. The model couplings are shown in Figure 4.

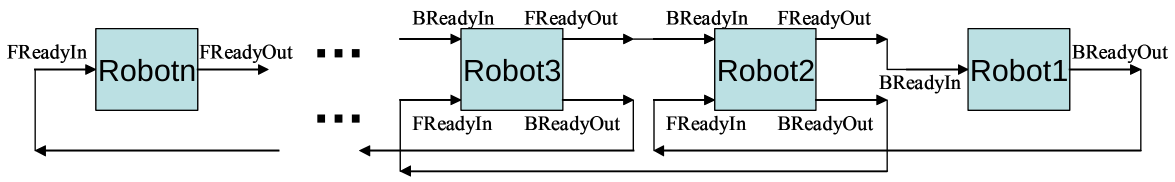


Figure 4: System model of the robotic convoy system

During the convoy, the *leader* robot (*Robot1* in Figure 4) decides the path of convoy. It moves straight forward if there is no obstacle ahead and turns if its infrared sensors indicate that there are obstacles ahead. All other robots conduct movement based on the moving parameters passed from their front robots and based on their own sensory inputs. Specifically, a robot first "predicts" where its front robot is and turns to that direction.

It then moves forward (or backward) to "catch" its front robot. After that it may go through an "adjust" process to make sure that it follows its front robot. This adjust process is necessary because noise and variance exist during a movement so a robot may not reach the desired position and direction after a movement. During adjustment, a robot "scans" around until it finds its front robot. Then it sends out a synchronization message to "inform" its front and back neighbors. Thus a robot actually goes through a basic "predict and turn—move—adjust—inform" routine. For example, a robot R_{i-1} will turn angle α_{i-1} to the direction of its front robot, move distance d_{i-1}, and then adjust itself with angle β_{i-1} to make sure it follows its front robot. Figure 5 shows these moving parameters.

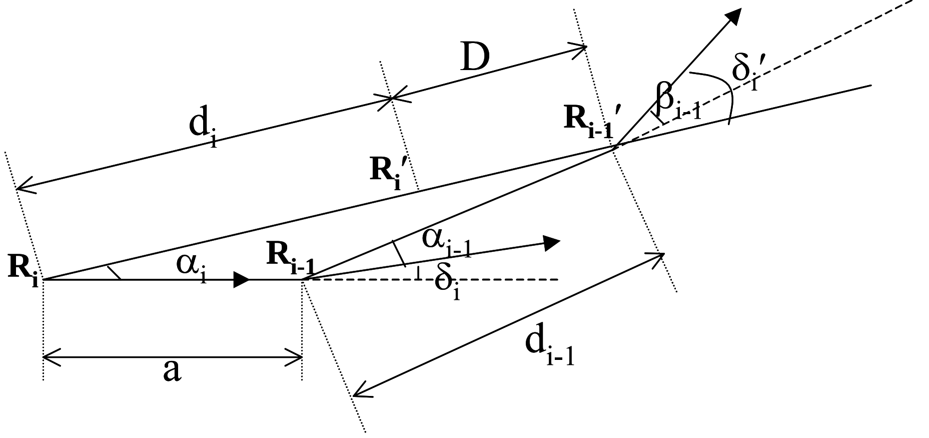


Figure 5: Moving parameters for robots' convoy

After the adjustment, R_{i-1} sends out a synchronization message to its neighbors. This synchronization message contains information of α_{i-1}, d_{i-1}, and β_{i-1}. Robot R_i then plans its movement based on the synchronization information as well as its own sensory data. This is shown by Figure 5 and formulated by formula (1)-(3). Among these formulas, δ_i is the angle (direction) difference between R_i and R_{i-1}; a is the distance between robot R_i and R_{i-1} and can be calculated from the robot's infrared sensor data and the size of the robot. Specifically, the turning angle α_i of R_i is calculated by formula (1); the moving distance d_i can be calculated from formula (2), where D is the desired distance between R_i and R_{i-1}. Then the new angle difference δ_i' between R_i and R_{i-1} is updated by formula (3), where β_i is the adjusting angle for R_i. This prepares for the next round of movements. We note that due to noise and variance, the δ_i' calculated from formula (3) will not be the actual angle difference between R_i and R_{i-1}. However, it seems that this error does not accumulate as time proceeds.

$$tg\alpha_i = \frac{d_{i-1} * \sin(\alpha_{i-1} + \delta_i)}{a + d_{i-1} * \cos(\alpha_{i-1} + \delta_i)} \qquad (1)$$

$$d_i = \frac{d_{i-1} * \sin(\alpha_{i-1} + \delta_i)}{\sin \alpha_i} - D \qquad (2)$$

$$\delta_i' = \delta_i + \alpha_{i-1} + \beta_{i-1} - \alpha_i - \beta_i \qquad (3)$$

The model of each robot is developed based on the subsumption architecture [2]. It has the *Avoid* model to avoid collisions with any objects; the *Convoy* model to control robot's movement based on the rules described above; and IR sensor and motor *Activities* to represent the IR sensor and motor interfaces of the robot. A detailed description of a similar model can be found at [15].

Models of IR Sensor and Motor

The motivation to design the above control logic, especially the "adjust" process, is because there exist significant uncertainty and inaccuracy in the movement of the mobile robots that we use. For example, when a robot is issued a command to move straight forward for a certain distance, it moves a different distant and also ends up with a different direction as compared to the original one. This is shown in Figure 6. Suppose the robot is in position *P1* and asked to move forward *D* distance, it will move *d* distance and stop at position *P2*. Furthermore, the direction of the robot is changed and there is an angle difference *a* between the new direction and original direction. To model this kind of motion uncertainty, two noise factors: distance noise factor (DNF) and angle noise factor (ANF) are developed and implemented using random numbers. The DNF is the ratio of the maximum distance variance as compared to the robot's desired moving distance. The ANF is the ratio of the maximum angle variance as compared to the robot's desired moving distance. For example, if the angle noise factor (ANF) is 0.1 and a robot moves forward 60 units distance, after its movement the robot will have maximum 6 degrees variance from its desired direction. Calculation based on the DNF is similar. Note that this model is a simplified treatment of motion uncertainty. In reality, a robot's movement trajectory is more likely to be the one as shown by the dotted line in Figure 6 (thus the robot will stop at position *P3*).

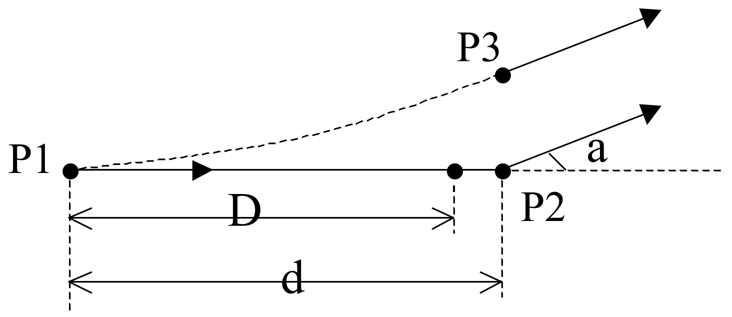


Figure 6: Model robots' motion inaccuracy

Due to the predominant uncertainty of the motion, less attention has been paid to model the inaccuracy of the IR sensors. The current sensor *abstractActivity* only takes into account the detection range (60 cm) of the IR sensor, i.e., robots can only detect objects within the range of 60cm. No noises are added to the IR sensory data. However, all IR sensory data are rounded to integer because this is the situation for real IR sensors.

Environment Model

The environment model is responsible to keep track of robots' movement and provides sensory dada when robots need it. Figure 7 shows the *Environment* model that is used for this example. This *Environment* model includes *TimeManager* models and the *SpaceManager* model. For each robot, there is a *TimeManager* corresponding to it. This *TimeManager* models the time for a robot to finish a movement. The *SpaceManager* models the moving space, including the dimension, shape and location of the field and the obstacles inside the field. It also updates and keeps track of robots' *(x,y)* positions and moving directions during simulation. Such tracking is needed to supply robots with the correct sensory data. To update a robot's position and direction, the environment model gets input message, which includes the moving parameters such as speed, distance, and noise factors (DNF and ANF), from the robot's motor *abstractActivity*. This message is first passed to the *TimeManager*, which calculates the moving time and delays the message for that period of time, and then to the *SpaceManager*, which calculates and updates the new position. Notice that in this example we have ignored the dynamics of a movement as we treat each movement as an atomic action so the positions and directions of robots are updated discretely.

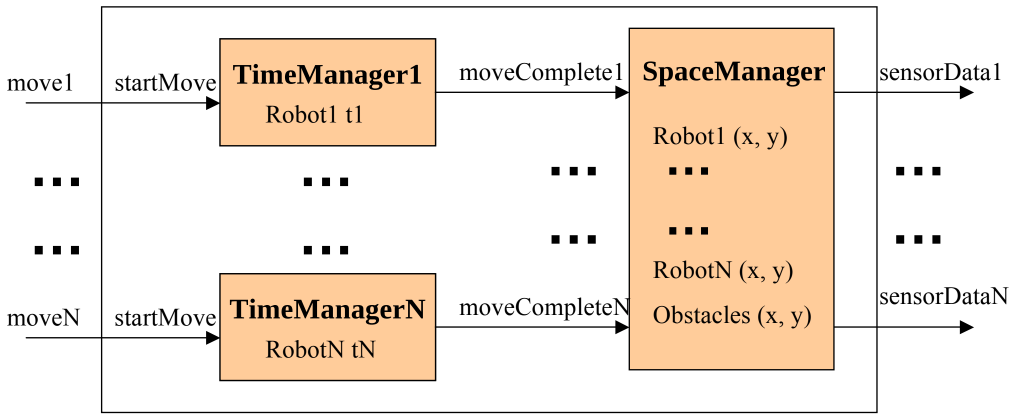
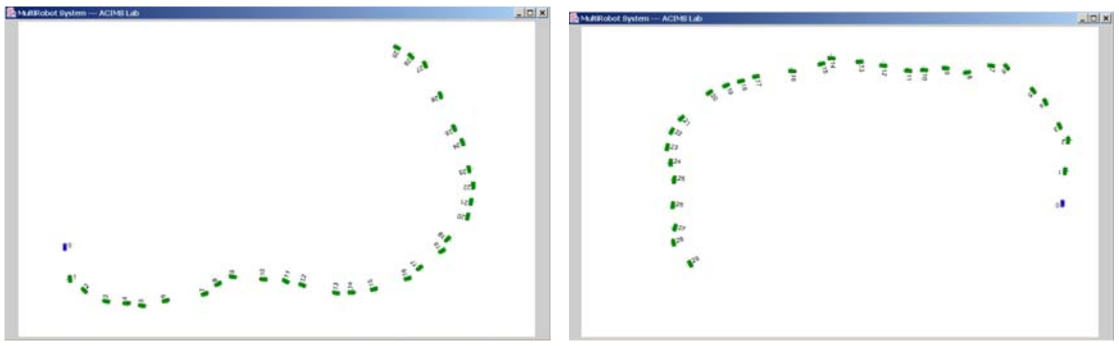


Figure 7: Environment model

With these models, simulations were run and a graphic user interface was developed to show robots' movements. Figure 8 shows two snapshots of a robotic convoy system with 30 robots within a field surrounded by walls (no obstacles inside). These simulations show that robots will not follow the exact track of the *leader* robot. But they are able to follow their immediate front robots, thus forming a coherent team from a global point of view.



(1) (2)

Figure 8: Snapshot of robots in motion

4.2 Measurement Metrics and Simulation Results

Follow the roadmap described in section 3.3, experimental frames can be developed to study this robotic convoy system. These experimental frames consist of three kinds of models. The *generator* models that represent the real world environment, e.g., the representation of motion inaccuracy in the convoy example – this puts out an error in the intended motion to an input command mimicking what the real floor does to the real robots. The *transducer* models that define the metrics used to measure team behavior. Some metrics are defined below. The *acceptor* models that specify the control employed on the experimentation, e.g., a way of stopping the experiment when the convoy breaks up entirely. Although detailed implementations of these experimental frames are still under development, a group of measurement metrics has been developed. Below we describe these metrics and present some simulation results based on them.

Convoy Speed and Number of Adjustment

The convoy speed of the robot team and the number of adjustment of each robot are among the most obvious results that can be obtained from simulation-based study. Both of them can be viewed as metrics for the system's performance. In fact, these two metrics are correlated to each other: the larger the number of adjustment, the slower the convoy speed. Since robots move in a coordinated way, we define the convoy speed as the speed of the leader robot. This can be calculated by dividing the moving distance by the simulation time. The number of adjustment can be obtained directly from each robot.

Formation Coherence

Due to motion inaccuracy, there exists difference between a robot's actual position, direction (angle) and its desired position and direction. This difference is affected by the variance of movement in real execution, which is modeled by DNF and ANF as described above. On the other hand, even though variance exists, this system can still conduct the convoy with some level of formation coherence. This is because an "adjust process" has been implemented that allows robots to adjust their positions/directions based on the feedback from its infrared sensors. Apparently the level of formation coherence is affected by the variance of movement. If this variance increases significantly, even though an adjust process exists, the system will fail to maintain its formation coherence.

To study this problem, we calculate each robot's position error (or position differences) under the effect of distance noise factor (DNF) and angle noise factor (ANF) and then derive the average position error of the whole team. This average is used as an indicator for the convoy system's formation coherence: the smaller the error is, the more coherent the convoy system is. Formula (4) – (7) shows how the average position error can be calculated. In these formulas, *D* is the desired distance between robots and *N* is the total number of robot. In case the formation is broken, saying robot R_i lose itself, $E_i(t)$ will increase continuously, making the average error *E(t)* increase too. Note that the desired position ($x_{i\text{-desired}}$, $y_{i\text{-desired}}$) of R_i is calculated from its front robot R_{i-1}'s position. Since relative values are used, systems with different formation shapes may result in the same position errors.

$$E_i(t) = \sqrt{(x_i(t) - x_{i-desired}(t))^2 + (y_i(t) - y_{i-desired}(t))^2} \quad (4)$$

$$x_{i-desired}(t) = x_{i-1}(t) - D * \cos\theta_{i-1}(t) \quad (5)$$

$$y_{i-desired}(t) = y_{i-1}(t) - D * \sin\theta_{i-1}(t) \quad (6)$$

$$E(t) = \sum E_i(t) \Big/ N \quad (7)$$

Figure 9 shows a sample average position error for a system with 30 robots, DNF=0.1, and ANF=0.08. The system starts with all robots at their desired positions. As simulation proceeds, the position error increases from 0 and then reaches a "stable" stage where the position error oscillates around an average value (35.7 in this example). Since the position error does not accumulate over time, we can say this system's formation coherence is maintained (a more formal definition of formation coherence is needed in order to reach any quantitative conclusions).

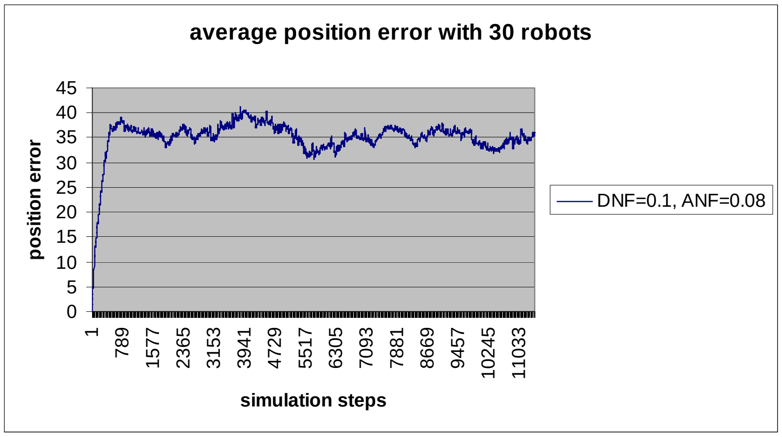


Figure 9: Average position error with 30 robots

Sensitivity

Since the formation coherence is affected by motion inaccuracy, sensitivity analysis is useful to study the system's robustness under the effect of different noise factors. To conduct sensitivity analysis, simulations with different DNFs and ANFs are run and then the corresponding position errors are calculated. The change of these position errors compared to the change of DNFs and ANFs indicates the sensitivity of the system. Figure 10 shows some average position errors for a system with 30 robots under the effect of three sets of DNF and ANF: set 1 has DNF = 0.04, ANF = 0.04; set 2 has DNF =0.1, ANF = 0.08; set 3 has DNF = 0.2, ANF = 0.1. For analysis purpose, we omit the "transient" stage when the simulations start.

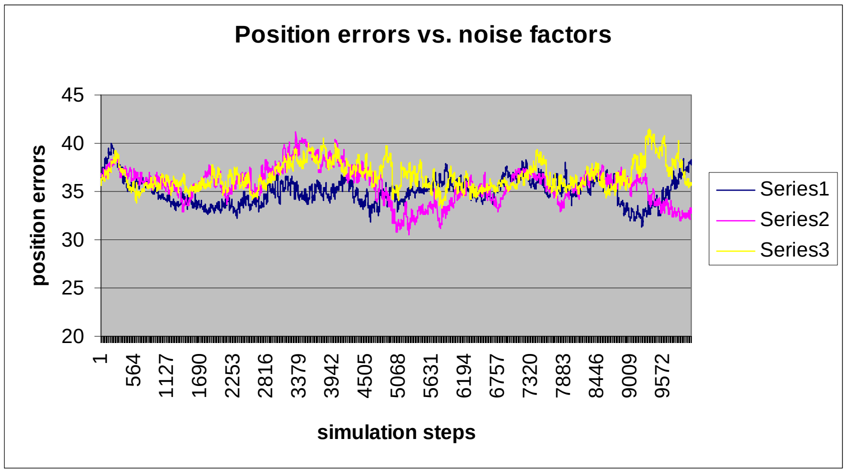


Figure 10: Average position errors vs. noise factors

Figure 10 shows that different noise factors result in different error patterns. However, these errors are maintained within a boundary (they do not accumulate as time proceeds). By calculating the average of them, we have average1 = 35.1, average2 =35.7, and average3 =36.6. These data show that the position error increases when the noise factor increases. However, this increase is insignificant as compared to the change of the noise factors. Although more analysis is needed to reach any quantitative conclusion, we can say that this system is insensitive to the noises as long as they are within a safe boundary. This is because the system impalements an adjust process that allows robots to adjust themselves based on the feedback from their IR sensors.

Scalability

Scalability refers to the ability of a system to maintain its quality as the scale of the system increases. To study scalability, we can change the number of robots and run simulation to see how that affects system's average position error (average over number of robots and over time). Figure 11 shows some position errors when the number of robots equals to 10, 20, 30, and 40 (with DNF =0.1 and ANF =0.08). It shows that the average position error increases as the number of robot increases. If this trend holds true with more robots, the system is not scalable in the sense that it will eventually break as more robots are added into the system.

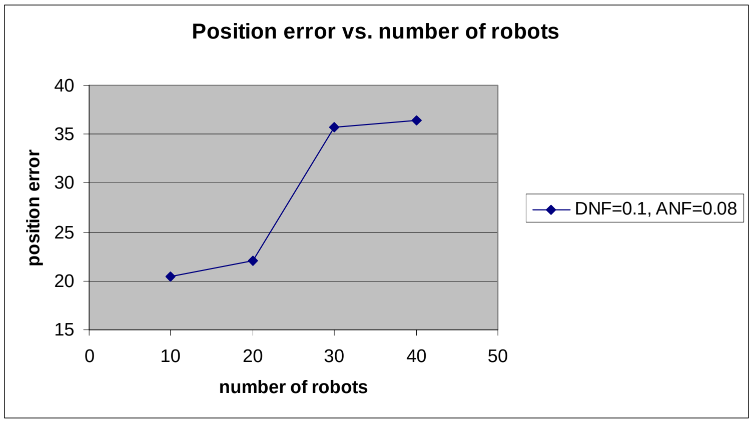


Figure 11: Average position errors vs. number of robots

4.3 An Experiment Using Robot-in-the-Loop Simulation

The data presented in the last section are from simulations that do not involve real robots. Experiments and analysis can also be conducted by using robot-in-the-loop simulation. Figure 12 shows an experimental setup that includes two real robots, *Robot_k* and *Robot_k+1*. These two robots neighbor each other and are physically placed on a floor that is free of obstacles. They are initially lined up so one follows the other. These two robots belongs to a convoy team and stay in the middle of it. The rest of the team is composed of robot models. One interesting aspect of this experiment is that these two robots represent different situations from the configuration point of view. The front real robot needs to follow *Robot_k-1*, which is a robot model simulated on computer. The back real robot needs to follow *Robot_k*, the front real robot. Thus in this experiment, *Robot_k* (the front real robot) is equipped with virtual IR sensor to gets sensory input from the virtual environment, while *Robot_k+1* uses its real IR sensor to track a real robot in the physical environment. Both of them are equipped with HIL Motors to move in the physical environment and to synchronize with the virtual environment. All other robot models use virtual IR sensors and virtual motors.

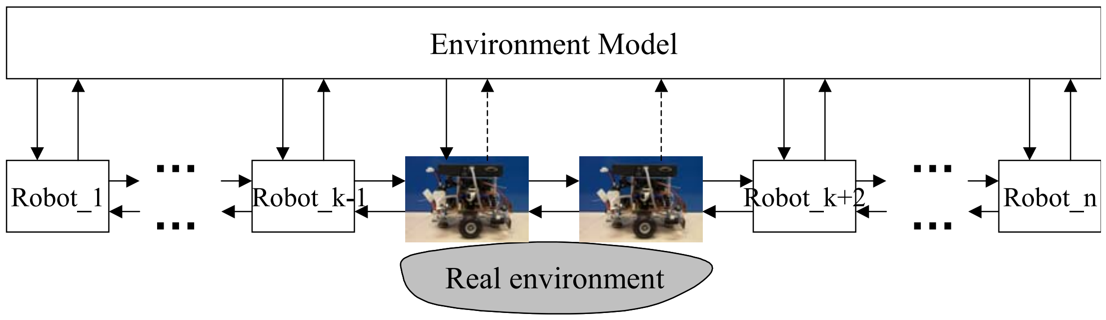


Figure 12: A robot-in-the-loop simulation setup

By running robot-in-the-loop simulation with this setup, several results can be collected and analyzed. For example, we can check if the back real robot really follows its front robot in the physical environment (Assuming they are able to do it in the simulation run on computer). If these two robots lose each other in robot-in-the-loop simulation, it means the robot models (either the decision-making model or the sensor/actuator models) are not good enough and need to be improved. Notice that in this case, the problems can be discovered by using only several real robots, instead of the entire real robot team. With robot-in-the-loop simulation, we can also compare some simulation results with those from the conventional simulation, and then use this information to refine the robot models under development. For example, for the experiment shown in Figure 12, we can measure and calculate the back real robot's position errors (based on the location of its front robot in the physical environment), and then compare it with the one collected from the conventional simulation. This information will be useful for analyzing and refining the model under development.

Quantitative results from this robot-in-the-loop simulation are still under collection and analysis. In the meantime, a movie [19] was recorded to qualitatively show how the two real robots work together with robot models. In this movie, four robots are used, among which the second and third ones are real robots. Other configurations are similar to Figure 12 described above. This movie includes two windows. One window shows the movements of two real robots. The other window shows the movements of simulated robot models, among which the second and third models are the counterparts of two real robots. Thus the second and third robot models' movements are synchronized with the two real robots' movements. For example, as can be seen in the movie, when a real robot moves backward (because it is too close to its front robot), the counterpart of this real robot in the simulation window moves backward too. This movie clearly demonstrates the coordination between the real robots and robot models. It also shows that the behaviors of these two real robots are comparable to those observed in conventional simulation.

5. CONCLUSION AND FUTURE WORK

This paper presented a hybrid approach of simulation that allows real robots as well as virtual ones to be studied together in a simulation-based virtual environment. The simulation architecture was described and an incremental study process, integrated with experimental frames, was proposed. An illustrative example of robotic convoy is described and some simulation results and a recorded movie were provided. The simulation data presented in this paper are from conventional simulation only. However, as a next step of this research, quantitative results from robot-in-the-loop simulation will be collected and analyzed. Comparing the results from conventional simulation and robot-in-the-loop simulation will reveal any potential problems of this simulation architecture and provide valuable information to improve it. Other future work of this research includes developing and implementing experimental frames for the robotic convoy example, and adding more complexities to the system to check if the simulation architecture can handle complex situations. This simulation-based virtual environment was originally developed in the context of mobile robot applications. Thus another future research task is to explore how it can be applied to other robotic applications such as object transportation, and material handling.

This paper has mainly focused on the architecture aspect of the simulation-based virtual environment. While an example is provided to illustrate some of the concepts, we believe the impact of this research reaches beyond the scope of this illustrative example. As the technology of robotic systems advance rapidly, systematic development methods and integrative development environments play more and more important roles in handling the complexity of these systems. The model continuity methodology and this simulation-based virtual environment promise such a systematic way to develop cooperative robotic systems.

6. ACKNOWLEDGEMENT

This research was supported in part by NSF grant DMI- 0122227, “DEVS as a Formal Framework for Scalable Enterprise Systems.”

7. REFERENCE

[1] T. Balch, Taxonomies of Multirobot Task and Reward, Robot Teams, Edited by Balch, T., and Parker L.E., A K Peters, 2002

[2] R. A. Brooks, A Robust Layered Control System For A Mobile Robot, IEEE Journal Of Robotics And Automation, RA-2, April. pp. 14-23, March 1986

[3] B. Browning, E. Tryzelaar, ÜberSim: a multi-robot simulator for robot soccer, Proceedings of the second international joint conference on Autonomous agents and multiagent systems, 2003

[4] T. K. Capin, I. S. Pandzic, N. Magnetat-Thalmann, and D. Thalmann, Avatars in Networked Virtual Environments, Wiley, 1999.

[5] L. Chaimowicz, M. Campos, and V. Kumar, Simulating Loosely and Tightly Coupled Multi-Robot Cooperation. In Anais do V Simp_osio Brasileiro de Automa_c~ao Inteligente (SBAI), Canela, RS, Brazil, September 2001.

[6] Y. K. Cho, RTDEVS/CORBA: A Distributed Object Computing Environment For Simulation-Based Design Of Real-Time Discrete Event Systems. Ph.D. thesis, University of Arizona, Tucson, AZ 2001

[7] Y. K. Cho, X. Hu, and B. P. Zeigler, The RTDEVS/CORBA Environment for Simulation-Based Design Of Distributed Real-Time Systems, Simulation: Transactions of The Society for Modeling and Simulation International, 2003, Volume 79, Number 4

[8] DEVS-Java Reference Guide, www.acims.arizona.edu

[9] K. Dixon, J. Dolan, W. Huang, G. Paredis, P. Khosla, RAVE: a real and virtual environment for multiple mobile robot systems, Intelligent Robots and Systems, 1999. IROS '99. Proceedings. 1999 IEEE/RSJ International Conference on , Volume: 3 , 17-21 Oct. 1999

[10] G. Dudek, M. Jenkin, and E. Milios, A Taxonomy of Multirobot Systems, Robot Teams, Edited by Balch, T., and Parker L.E., A K Peters, 2002

[11] K. Goldberg, The Robot in the Garden, Telerobotics and Telepistemology in the Age of the Internet, MIT Press, 2000.

[12] J.S. Hong, and T.G. Kim, Real-time Discrete Event System Specification Formalism for Seamless Real-time Software Development, Discrete Event Dynamic Systems: Theory and Applications, vol. 7, pp.355-375, 1997.

[13] A. Howard, N. Koenig, Gazebo, Version 0.3.0 User Manual, University of Southern California technique report, 2004

[14] X. Hu, A Simulation-based Software Development Methodology for Distributed Real-time Systems, Ph.D. Dissertation, University of Arizona, 2004

[15] X. Hu, and B. P. Zeigler, Model Continuity to Support Software Development for Distributed Robotic Systems: a Team Formation Example, Journal of Intelligent & Robotic Systems, Theory & Application,pp. 71-87, January, 2004

[16] X. Hu, and B. P. Zeigler, Model Continuity in the Design of Dynamic Distributed Real-Time Systems, accepted by IEEE Transactions On Systems, Man And Cybernetics— Part A: Systems And Humans, in May 2004

[17] K. Komoriya, K. Tani, Utilization of the virtual environment system for autonomous control of mobile robots, Intelligent Motion Control, 1990. Proceedings of the IEEE International Workshop on, Volume: 2, 20-22 August 1990

[18] O. Michel, Webots: Professional Mobile Robot Simulation, International Journal of Advanced Robotic Systems, Vol. 1, Num. 1, pages 39-42, 2004

[19] Movie http://www.cs.gsu.edu/~cscxlh/ril2.html, Macromedia Flash (SWF) movie file, playable using Internet Explore

[20] Y. Ohta, and H. Tamura, Mixed Reality, Merging Real and Virtual Worlds, Springer-Verlag, 1999.

[21] J. Peipelman, N. Alvarez, K. Galinet, R. Olmos, 498 A & B Technical Report. Department of Electrical and Computer Engineering, University of Arizona, 2002

[22] Robot link: http://www.cs.gsu.edu/~cscxlh/research.htm

[23] C. Sayers, Remote Control Robotics, Springer-Verlag, 1998.

[24] S. C. Shaw, Real-time Systems and Software, John Wiley & Sons, 2001

[25] R. Vaughan, Stage: a multiple robot simulator. Technical Report IRIS-00-394, Institute for Robotics and Intelligent Systems, School of Engineering, University of Southern California, 2000.

[26] J. Wang, Methodology and design principles for a generic simulation platform for distributed robotic system experimentation and development. Systems, Man, and Cybernetics, 1997. Computational Cybernetics and Simulation., 1997 IEEE International Conference on, Volume: 2, 1997 Page(s): 1245 -1250 vol.2

[27] B. P. Zeigler, Y. Moon, D. Kim, J. G. Kim: DEVS-C++: A High Performance Modelling and Simulation Environment. *HICSS* (1) 1996: 350-359

[28] B. P. Zeigler, H. Preahofer, T. G. Kim, Theory of Modeling and Simulation, New York, NY, Academic Press, 2000.