

AN INTEGRATED MULTI-ROBOT TEST BED TO SUPPORT INCREMENTAL SIMULATION-BASED DESIGN

*Ehsan Azarnasab**

Georgia State University
Department of computer Science
Atlanta, GA, 30303

Xiaolin Hu†

Georgia State University
Department of computer Science
Atlanta, GA, 30303

ABSTRACT

Design of complex systems calls for systematic methods as well as supporting environments to experiment with, understand, and evaluate them. For robotics systems that include a large number of robots, an integrated simulation-based design environment will greatly simplify the transitions from robot model experiment to real robot deployment and execution. This paper presents the development of an integrated multi-robot test-bed to support an incremental simulation-based design methodology that includes conventional simulation, robot-in-the-loop simulation, and real robot experiment. We describe the hardware, software, and system architecture of this integrated test bed, and present a multi-robot dynamic team formation example to demonstrate the usage of this platform along the different stages of the design process.

Index Terms— Robot-in-the-Loop simulation, DEVS, Simulation based design, Mobile robots, Pattern recognition, Image processing

1. INTRODUCTION

A multi-robot system consists of multiple robots that coordinate with each other to finish a common task. Design of multi-robot systems is a challenging task due to the factors such as dynamic environment, real time processing, hardware constraint, and complex cooperation and coordination requirement. This is especially true for large scale robotic systems that have a large number, e.g., up to hundreds, of robots and need to accomplish complex tasks. The increasing complexity of cooperative robotic systems calls for systematic methods as well as supporting environments to experiment with, understand, and evaluate them.

Modeling and simulation have been widely applied in supporting multi-robot system development. In previous work, we have developed a model continuity design methodology [1] and a Robot-In-the-Loop (RIL) simulation method that allows

real robots to be experimented in a simulation-based virtual environment containing many robot models [1]. RIL simulation is carried out by replacing some robot models in a multi-robot system with real robots. These real robots use a combination of virtual and real sensors/actuators. For example, a real mobile robot may use its virtual IR sensors to get sensory inputs from a virtual environment and, as a response to the decision-making of these inputs, use its real motors to move the robot in the physical environment. The real robot can also sense other robot models simulated on computers and communicate/coordinate with them. Setups like these allow designers to test different aspects of the robot system in a flexible yet controlled manner. Including real robots in a simulation-based study has several advantages from the design point of view. First, RIL simulation brings simulation-based study one-step closer to the reality. Measurement results from RIL simulation can be compared with those from conventional simulation. Such comparisons will provide useful feedbacks to the designers, e.g., indicating that a robot model may not model the real robot's movement very well. Meanwhile, by using real robots, RIL simulation increases designers confidence about how the final real system is going to work. Note that in both of these two cases, RIL simulation allows the designers to use only several, instead of all, real robots to gain the above knowledge. For large-scale cooperative robotic systems that include hundreds of robots, RIL simulation thus makes it possible to conduct system-wide tests and measurements without waiting for all real robots to be available. This capability is especially useful for large-scale cooperative robotic systems whose complexity and scalability severely limit experimentations in a physical environment using real robots. Therefore, using the idea of RIL simulation we can test an incomplete system by replacing the unavailable parts with the models. This is also useful in testing the combat scenarios when the model of other competitors is tested against a team of robots.

The intermediate stage of RIL simulation makes it possible to add RIL simulation to conventional simulation and real system experiment to form an incremental simulation-based design process. The design process will start from conven-

*At University of Utah, department of Electrical and Computer Engineering

†At Georgia State University, department of computer science

tional simulation where all robots are models and simulated on a centralized computer, and then transition to robot-in-the-loop simulation that includes combined real and virtual robots, and finally to real system experiment where all robots are real and tested in the real field just like how the final system should be. In one sense, RIL based design can be compared with model based system control, that a model is generated for part of the system which can improve over successive information of the sensors.

To support the incremental simulation-based design process described above, an effective multi-robot test bed needs to be developed. This test bed should support all stages, including conventional simulation, robot-in-the-loop simulation, and real robot execution, of the design process in a coherent way, and provide an integrated interface for all the three stages and enable smooth transitions between them. This paper presents the development of such an integrated multi-robot test bed based on Khepra mobile robots, which have been widely used in many multi-robot research and education projects. We describe the hardware, software, and system architecture of this integrated test bed, and present a multi-robot dynamical team formation example to demonstrate the usage of this test bed. Modeling and simulation of virtual robots and the environment are based on Discrete Event System Specification (DEVS) [2], and specifically the DEVSJAVA [3] simulation package. Detailed descriptions about how robot models are developed and how simulations (including conventional simulation and robot in the loop simulation) are supported can be found in [1, 4] and thus are omitted here.

In the first part of this text, the related work, the system architecture (including the image processing part in software) and the hardware setup are discussed. Then the effect of robot-in-the-loop stepwise system integration is introduced.

2. RELATED WORK

Multi robot systems and robot swarms [5] have been an active research area in robotics and related topics. The control aspect and path planning based on robot models are widely researched and exploited in car factories and automation industry, mainly for robotic arms. For mobile robots, the problem is not often predefined which makes it more dynamic. Some of the mobile robot system simulations are pure simulations (like Webots [6] originally for Khepra robot simulation), some work with real robots and some are hybrid of real and virtual objects [1]. In [1], a DEVS based simulation is discussed. Another hybrid framework for real and virtual environment (RAVE) was proposed [7] which addressed the benefit of the virtual augmentation to test virtual sensors, and virtual robots when the number of real robots is limited. RAVE is based on libraries, servers and user interfaces to perform different scenarios. The work of [8] puts the idea of mixed reality in the context of distributed robotic system

(with objects responding to external events) with the concepts of virtual sensors, and suggests this approach when the cost of robots is high. Modeling the sensors can be more detailed to handle uncertainty [9].

Another form of hybrid simulation is projective virtual reality [10, 11] which projects the virtual world into reality. The concept of mixed reality is utilized in Virtual Reality (VR), with many applications for example in flight simulators [12], bio-informatics [13], realtime weather forecasting, and mobile robotic test-beds [10, 11, 14, 15, 16, 17, 1]. The extensive control capabilities of Khepra robots when combined with a good localization method (like image processing with external camera) is suitable for this kind of virtual and reality integration as in KhepOnTheWeb [14, 18]. Gracanin's virtual reality test-bed [15] adds virtual objects to the user interface (virtual environment and robots) and distributed users can control the same robot in a virtual world. This research also emphasizes the importance of the network resources, and implements a distributed control protocol. In the current work we realized the importance of resource management (like networking) when more robots added to the complexity of the system.

3. SYSTEM ARCHITECTURE

The goal of this project is to design a collaborative system consisting of many agents, to experiment with the idea of *agent-in-the-loop* system design, and here when we refer to agents it means both real and virtual robots. All agents are treated the same, while they can have different interfaces to other entities. Agents, regardless of being real or virtual entities, are DEVS models each with a set of behaviors in a behavior network. Having the same code for agents, in one hand makes the design simpler and makes it possible to go from simulation to implementation easier, on the other hand it requires homogeneity of the agents. To tackle the inherent heterogeneity of real entities the control model is made more robust and flexible enough not to be bound to certain parameters. The control unit performs actions by the developed software.

3.1. Software

When the software is running, control panel takes control of other components (Fig. 1). User can change parameters related to vision, localization and navigation. User can also manually navigate real robots or change the speed parameters. In addition, the user can switch to central or distributed automatic convoy *mode* and start or stop the real-time simulation of the system. In this design, the user can interact only with the *control panel* through its Graphical User Interface (GUI), therefore the internal architecture of the application is transparent from user point of view. Localization of

real robots is performed by image processing, then the simulated system model of agents (AgentSys in Fig. 1) is updated by new external events of vision activity. Finally, based on this new events, the behavior network of each agent decides the next simple movement which is then transmitted to corresponding agents by appropriate interfaces. The AgentSys component is modeled in DEVS [2] and uses DEVSTJava [3] as the simulation engine.

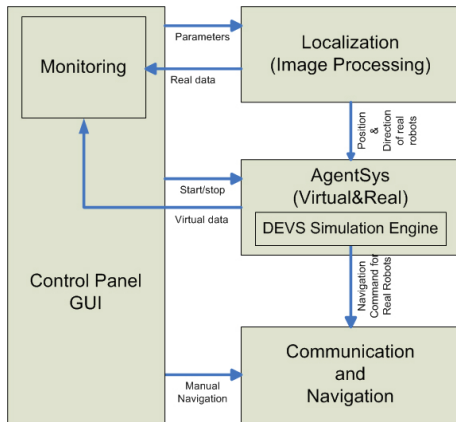


Fig. 1. System architecture

The user can monitor the current state of the system (agents' positions and selected behaviors) in the same GUI. The localization unit informs the monitoring component of the real robots, while the DEVS simulation core (AgentSys) adds the virtual robots information. The GUI is designed so that it simplifies the interaction of user and simulation software, making it suitable for teaching concepts of robotics, behavior selection, DEVS modeling and simulation.

The AgentSys model consists of the Environment model and the Agent models; The environment model is a DEVS atomic model that models the physical environment of the robot system (Fig. 2) and information of all robots (real and virtual), such as their dimensions, positions, and moving directions. The environment gets the result of the vision activity and dispatches data to the related agents. It also builds the virtual layer on top of the real layer in the control panel GUI. This virtual layer comprises the virtual robots (red circles in Fig. 5), rectangular boundary of the virtual environment (green rectangle in Fig. 5) and the names of current behaviors of agents. Some of these virtual data can be turned off by the user. Furthermore, vision activity is a DEVS activity that belongs to the real-time DEVS and closes the loop of the control system. For any other localization sensor if added to the system, an activity regarding the new sensor is needed. In the case of multiple input sensors, the result of all activities should be fused to find knowledge about the system.

Agents, regardless of being real or virtual entities, are DEVS models each with a set of behaviors in a behavior network as the decision making part. Robot team formation (in

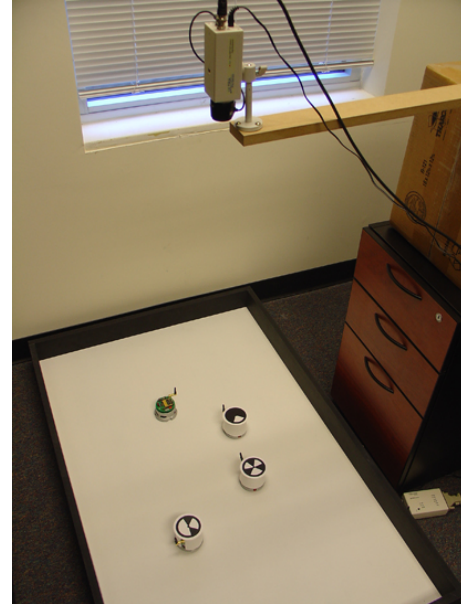


Fig. 2. System hardware, the robots, and environment consisting the overhead camera and the rectangular filed

central convoy) requires sending navigation commands to the robots, also the distributed convoy may broadcast the information about the real robots to them so that their behaviors change by new information. As a result, the agents access the environment by appropriate interfaces. Virtual agents get access to the virtual environment, while real robots interact with real environment. Therefore in the central team formation, real and virtual interfaces are implemented and by automating the robots in distributed convoy the virtual interface is replaced with the real robot on-board software. For an example, in central convoy when a behavior of an agent is activated the navigation commands are sent to the environment interface of the agent. If the agent is a real robot, the navigation command is transmitted over the communication link, but if the agent is virtual, the command is simulated and thus changes the position or heading of the virtual robot.

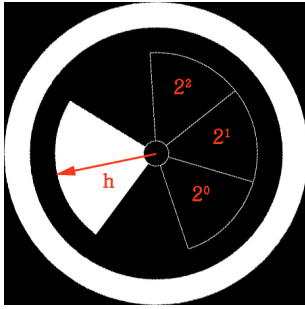
3.2. Hardware

Images of the field are taken by an overhead camera located so that the grabbed picture covers the entire filed on which the robots move (Fig. 2). By this choice we avoided the extra processing inside each robots and also using available techniques of image processing we could find the location and direction of each robot accurately. This camera is installed on top of the platform where the robots are moving (1.2×1.5 square meters) in the height of about 1.5 meters. The robots are Khepra[®] robots with circular shape and 8 proximity sensors. Having the global view of the filed by the image of the field in hand, the sensors are modeled for the real robots

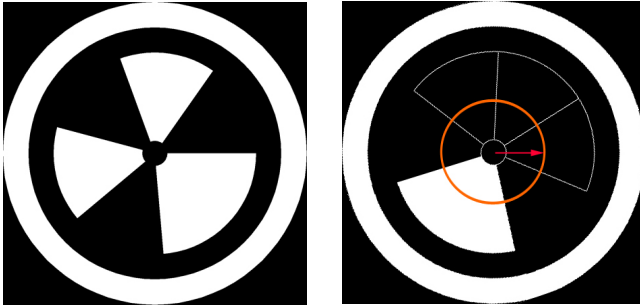
avoiding the extra need for filtering of the sensors. The image of the field is taken by the overhead camera then used for the localization.

4. LOCALIZATION AND VISION

Environment as a real-time atomic model runs a DEVS activity (called vision activity) to get the new position and heading of all real robots leading to vision based localization. Image processing is used to find information of real robots on the field. Agents use these information as if they are looking for other agent, thus image processing is basically the vision of agents. As we consider the vision in the first part to capture distances of the robots and informing them, one option would be to use vision on robots themselves to make the simulation more natural. However on-board image processing with all pixel data is time consuming and requires too much memory, so people come up with complicated algorithms to tackle this problem for real-time robotics [19].



(a) Pattern template



(b) 6UCP for ID = 6

(c) Sampling

Fig. 3. Circular pattern with 6 slots.

4.1. Pattern

To find robots more accurately, while not relying on robot processing abilities, we used a conventional method including an over-head camera to track the movement of the robots by finding the special patterns on top of them. We used a 6 angle circular pattern with color coding on the angular dimension of the pies (Fig. 3(a)). Figure 3(b) shows the designed 6 angle pattern for ID equal to 6. This patterns consist of 6 pie slices, one inner hole and two outer rings. To color code the pattern we use the *pattern template* scheme in Figure 3(a), in which the largest pie slice is always white as well as the outer ring, also the inner hole and two pie slices near the biggest pie are always black. Therefore by considering white as 1 and black as 0, three remaining pie slices can be used to identify 2^3 or 8 different IDs for robots.

Definition 4.1 A 6-angle-uniform-circular-pattern (6UCP) is defined by three vectors and one coding formula:

$$P_6 = \left\{ \begin{array}{l} \theta = [\theta_1, \theta_2, \dots, \theta_6] \\ \beta = [\beta_1, \beta_2, \dots, \beta_6] \\ \alpha = [\alpha_1, \alpha_2, \dots, \alpha_6] \\ \phi, r_1, r_2 \end{array} \right\}$$

where,

θ is a vector of angles of 6 pie slices. such that:

$$\sum_{i=1}^6 \theta_i = 360^\circ \quad (1)$$

$\beta_i \in \{0, 1\}$ for $i = 1, 2, 3, 4, 5, 6$. β is the color vector so that if $\beta_i=1$ the i^{th} pie slice is white otherwise it is black.

α is the vector of position exponents (α_i are natural numbers or zero, for $i = 1, 2, 3, 4, 5, 6$)

ϕ is the code (or ID) of the pattern such that:

$$\phi = \sum_{i=1}^6 \beta_i \times 2^{\alpha_i} \quad (2)$$

r_1 and r_2 are the radii of inner hole and inner ring respectively.

Obviously $0 < r_1 < r_2$. Also because this definition corresponds to the polar coordinate system then all angles are counted in counter clockwise (CCW).

Note that equation 1 is needed for the pattern to be periodic. Also because the data is coded only on the angular dimension of the polar coordinate system of the pattern. The pattern is uniform in magnitude dimension (towards the radius of the circle). Based on this definition, one possible instance for the 6UCP with which we made our pattern (and is shown in Fig. 3) is as follows:

$$\begin{aligned}
\theta &= [85, 55, 55, 55, 55, 55] = 55 \times [1.5, 1, 1, 1, 1, 1] \\
\beta &= [1, 0, x_1, x_2, x_3, 0] \\
\alpha &= [0, 0, 1, 2, 3, 0]
\end{aligned} \tag{3}$$

Note that x_1, x_2 and x_3 refer to the three labeled pies in Fig. 3, they can be 0 representing black or 1 representing white. Also for convenience the biggest angle in the pattern should be the first in vector θ . The color corresponding to this angle is always white (because $\beta_1 = 1$), this results in an offset in building the codes therefore the code sequence begins with 1 ($\phi \in \{1, 2, 3, 4, 5, 6, 7, 8\}$). One observation is that although this instance can describe the pattern we chose, it is not the only possible instance of 6UCP. Another observation is that if we shift all three vectors circularly we would have the same result, and it is due to the fact that if a robot turns it still will have the same ID. Therefore the algorithm to identify this pattern also should overlook the rotation. That is why a *Circular Cross Covariance* can detect this pattern.

4.2. Image Processing

The vision algorithm consist of the following steps; build the image, convert it to gray scale, enhance the contrast, find the center of objects by blob detection analyze the pattern and map the objects to robots. In practice, enhancing contrast by stretching the histogram of the image, proved to make the algorithm robust against the noise. In blob detection, for each line of the image and each pixel, if the pixel value is inside a certain threshold range, the algorithm traces the edge to mark the blob (while calculating measurements such as centroid of the blob) with a special color outside the threshold. The algorithm continues to find all blobs (found objects with certain circularity and size). The centroid is calculated by averaging the x and y coordinates of all of the pixels in the selection (here circular blobs), thus it is to some extent robust against lenses effect. Using normal camera and without preprocessing the image, the centroid measurement was better than finding the center by special color in the center.

Sampling starts with 0 degrees (in polar coordinate around the found centroid) and adds δt in each step. In building the ideal signals, we start with the β_1 and sampling also starts at the beginning of this segment (we assumed the pattern has not turned around the center). In addition, there is no noise in the ideal signal. However, the pattern might be rotated (Fig. 3(c)) causing the signal to be shifted. In practice some noise is also added to the signal. To improve the signal being sampled the brightness of captured frames can be changed in vision setting. Now the identification method would be this; the test signal should be compared with all available ideal signals (in the bank) to find the one with more correlation. The lag at which the correlation gets its maximum value, is directly related to the heading of the robot pattern (red arrow starting from the center at Fig. 3(a)). The mutual similarity of an ID

with another ID is defined as the circular cross covariance of the sampled signals and ideal signals of them.

$$CX(d) = \frac{\sum_{i=1}^N [(x(i) - m_x)(y(i-d) - m_y)]}{\sqrt{\sum_{i=1}^N (x(i) - m_x)^2} \times \sqrt{\sum_{i=1}^N (y(i) - m_y)^2}} \tag{4}$$

where N is the length of the vectors, also m_x and m_y are the mean of the vectors x and y respectively.

$$y(-k) = y(N - k) \tag{5}$$

$$\eta(x, y) = \max_d (CX(d)) \tag{6}$$

We can have up to 8 different markers in 6UCP but not all of them are generally in the field (not all of them are available markers). As a result we have m available markers ($1 \leq m \leq 8$) with associated IDs of $\{ID_1, \dots, ID_m\}$. Correlating each found object (totally n objects of $\{o_1, o_2, \dots, o_n\}$) with all the available patterns in the field, a table of similarity between objects and robot patterns is achieved. Table 1 shows this mapping for 5 found objects when only three robots are in the field.

	$ID_1 = 1$	$ID_2 = 3$	$ID_3 = 6$
o_1	0.27433464	0.3336813	0.40391427
o_2	0.6221739	0.93153024	0.16337222
o_3	0.5072501	0.19723846	0.9139368
o_4	0.3381496	0.47814554	0.36910614
o_5	0.9299889	0.65918684	0.51369625

Table 1. Object to ID similarity mapping for $m = 3$ and $n = 5$

In the final step of image processing we should assign each ID to an object (the reverse of the mapping). Note that we assumed all of these IDs exist in the field, then we should find the objects which are more likely to be a particular ID, while each object can be used only once. The localization results is applied directly in the central team formation robot models, while it is transmitted to the robots in the distributed version.

5. A MULTI-ROBOT TEAM FORMATION EXAMPLE

The decision making core of the agents (real and virtual robots) is a network behaviors based on mutual inhibition behavior choice and is modeled in DEVS. Detailed description of system and its control models can be found at [4]. Here we focus on the design process aspect and give an example. We first give an overview of the system and its task, and then show

the different stages of the simulation-based design process as described before.

To help the convoy to form, the agents (including real and virtual robots) are assigned IDs consecutively from 1 to N , where N is the total number of agents. A formed team is a convoy in which the k^{th} agent (R_k) should follow the agent R_{k-1} for $k \in \{2, 3, \dots, N\}$. The final team would be $\{R_1, R_2, \dots, R_N\}$. By removing the ordering of the agents, emergent behaviors also can be investigated in the future research. Each agent should *follow* the next agent or *search* for it, while *waiting* for the previous agent. The team formation is dynamic; during the process; sub-teams may form ($\{R_l, R_{l+1}, \dots, R_m\}$ that $1 \leq l < m \leq N$) that later on this chain is broken because of other phenomena. For example another agent which does not belong to this sub-team may try to cross the line between R_o and R_p ($l < o < p \leq m$), the chain is widened at that point not to *collide* with the crossing agent, therefore the agent R_o may decide not to wait for R_p anymore and continue its convoy. This active environment especially when real objects are in the simulation, make the predefined path planning for team formation very hard.

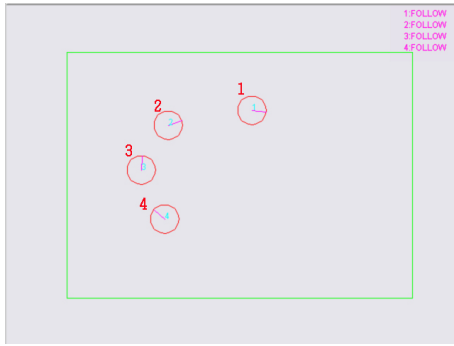


Fig. 4. Pure simulation of 4 agents.

Based on Robot-in-the-loop methodology, the design process starts with pure simulation. Because this step does not involve real robots, the simulation is run as fast as it can. The speed we buy in this way helps to find the logical design flaws and the conflicts very soon, which is one of the primary use of simulation in conventional simulation-based design without exploiting the model continuity. The result of this step yields a statistically correct architecture with effect of accuracy and reliability of the system in the long run or under different conditions. Because of the fast simulation, the general attitude of the system is investigated and the final goal (which may take time in reality) is tested if it is achieved at all or not. For example for the robot convoy problem it is tested if the agents finally form a team (Fig. 4) under different initial conditions such as different size, number, position and heading for the agents.

Fig. 5 shows a team formation example as an application of the robot-in-the-loop simulation. In this stage of the design

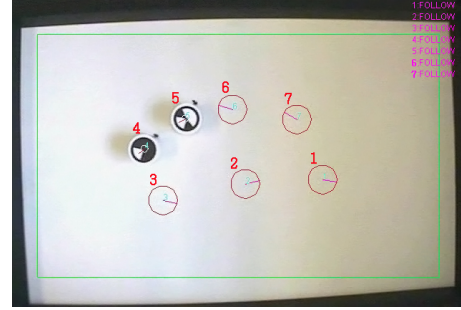


Fig. 5. Two real robots in the loop, totally 8 agents.

two real robots (R_4 and R_5) are added to the convoy and the team is already formed. All robots are in following behavior. The environment boundary is the green rectangle in this figure which robots avoid as the virtual walls. Another feasible case study for this system design would be to have real and virtual robots compete with each other in two different teams. The part with mixed reality is simulated in both central and distributed control. In distributed control, only the virtual robots are controlled by the main software, while the real robots have their own decision making. In central control, all agents are controlled centrally. Finally, a system of only real robots is tested (Fig. 6).

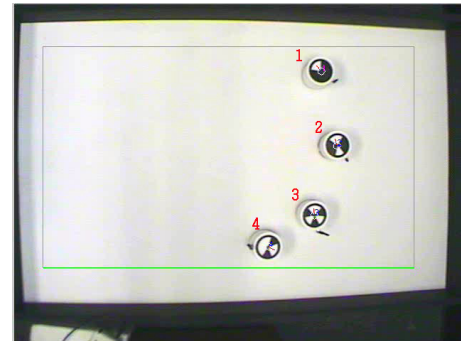


Fig. 6. Four real robots in a formed team.

6. CONCLUSION AND FUTURE WORK

How to design and implement a multi agent system is a complex task, that requires a precise modeling of the system and a suitable decision making part. We used the simulation (real-time and non real-time) to test and refine the system model, in order to improve the decision making part. To further simplify this job, we used robot-in-the-loop simulation methodology, and increased the complexity of the system step by step. As we chose the mutual inhibition based behavior mechanism for our decision making part, the simple behaviors are built by each step of the simulation. These behaviors are modeled (in DEVS) as a behavior network which decides the next naviga-

tional movement of the robot. The robot team formation, as a case study for robot-in-the-loop simulation, is implemented with a central computer (running the real-time simulation). The next phase of the project comprises the distributed team formation in which the behaviors (which are already defined) are programmed on each robot individually to see the effect of the step-wise design pattern. In both parts of the project, the robots each is assigned an ID which determines its position in the convoy. The performance of this system design and comparing the simulation and real experiments is another area we are currently working on.

7. REFERENCES

- [1] X. Hu and B.P. Zeigler, "A Simulation-Based Virtual Environment to Study Cooperative Robotic Systems," *Integrated Computer-Aided Engineering (ICAE)*, vol. 12, no. 4, pp. 353367, November 2005.
- [2] Bertrand P. Zeigler, Herbert Praehofer, and Tag Gon Kim, *Theory of Modeling and Simulation: Integrating Discrete Event and Continuous Complex Dynamic Systems*, Academic Press, 2 edition, 2000.
- [3] Bertrand P. Zeigler and Hessam S. Sarjoughian, "Introduction to DEVS Modeling and Simulation with JAVA: Developing Component-Based Simulation Models", jan 2005.
- [4] Ehsan Azar, "Robot-in-the-loop simulation to support multi-robot system development: a dynamic team formation example," M.sc., Georgia State University, Dec 2006.
- [5] James McLurkin, "Stupid Robot Tricks: A Behavior-Based Distributed Algorithm Library for Programming Swarms of Robots," M.sc., M.I.T., May 2004.
- [6] O. Michel, "Webots: Professional Mobile Robot Simulation," *International Journal of Advanced Robotic Systems*, vol. 1, no. 1, pp. 39–42, 2004.
- [7] K. Dixon, J. Dolan, W. Huang, and G. Paredis and P. Khosla, "Rave: a real and virtual environment for multiple mobile robot systems, Intelligent Robots and Systems," *IROS '99. Proceedings. 1999 IEEE/RSJ International Conference on AAI Mobile Robot Workshop*, vol. 3, Oct 1999.
- [8] J. Wang, "Methodology and design principles for a generic simulation platform for distributed robotic system experimentation and development.," *IEEE International Conference on Computational Cybernetics and Simulation*, vol. 2, pp. 1245–1250, 1997.
- [9] Lei Yang, Xinxin Yang, and Kezhong He, "The research on mobile robot simulation and visualization under virtual reality," *Proc. of International Conference on The research on Information, Communications and Signal Processing, 1997. ICICS.*, vol. 1, pp. 390–396, September 1997.
- [10] E. Freund and J. Rossmann, "How to control a multi-robot system by means of projective virtual reality," *Proc. of 8th International Conference on Advanced Robotics (ICAR)*, pp. 759–764, 1997.
- [11] Freund E., Hahn D.L., and Rossmann J., "Cooperative control of robots as a basis for projective virtual reality," *Proc. of 8th International Conference on Advanced Robotics (ICAR)*, pp. 753–758, 1997.
- [12] R.G. Menendez and J.E. Bernard, "Flight simulation in synthetic environments," *Proc. of Digital Avionics Systems Conferences (DASC)*, vol. 1, pp. 2A5/1–2A5/6, 2000.
- [13] E. Moritz and J. Meyer, "Interactive 3D protein structure visualization using virtual reality," *Proc. Bioinformatics and Bioengineering (BIBE)*, May 2004.
- [14] O. Michel, P. Saucy, and F. Mondada, "KhepOnTheWeb: An experimental demonstrator in telerobotics and virtual reality," *Proc. of International Conference on Virtual Systems and MultiMedia (VSMM)*, pp. 90–98, 1997.
- [15] D. Gracanin, K. Matijasevic, N.C. Tsourveloudis, and K.P. Valavanis, "Virtual reality testbed for mobile robots," *Proc. of IEEE International Symposium on Industrial Electronics (ISIE)*, vol. 1, pp. 293–297, 1999.
- [16] Cao Bailin, G.I. Dodds, and G.W. Irwin, "An event driven virtual reality system for planning and control of multiple robots," *Proc. of IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, vol. 2, pp. 1161–1166, 1999.
- [17] E. Freund and J. Rossmann, "Projective virtual reality: bridging the gap between virtual reality and robotics," *IEEE Transactions on Robotics and Automation*, vol. 15, no. 3, pp. 411–422, 1999.
- [18] Saucy P. and Mondada F., "KhepOnTheWeb: open access to a mobile robot on the internet," *IEEE Robotics and Automation Magazine*, vol. 7, no. 1, pp. 41–47, March 2000.
- [19] Bryan Scotney, Sonya Coleman, and Dermot Kerr, "A Graph Theoretic Approach to Direct Processing of Sparse Unwarped Panoramic Images," *IEEE International conference on Image Processing*, Oct. 2006.