5-3-2007

Robot-In-The-Loop Simulation to Support Multi-Robot System Development: A Dynamic Team Formation Example

Ehsan Azarnasab

ROBOT-IN-THE-LOOP SIMULATION TO SUPPORT MULTI-ROBOT SYSTEM DEVELOPMENT: A DYNAMIC TEAM FORMATION EXAMPLE

by

Ehsan Azarnasab

Under the direction of Xiaolin Hu

ABSTRACT

Modeling and simulation provides a powerful technology for engineers and managers to understand, design, and evaluate a system under development. Traditionally, simulation is only used in early stages of a system design. However, with the advances of hardware and software technology, it is now possible to extend simulation to late stages for supporting a full life cycle simulation-based development. Robot-in-the-loop simulation, where real robots work together with virtual ones, has been developed to support such a development process to bridge the gap between simulation and reality.

INDEX WORDS: Robot-in-the-Loop system design, DEVS, Real-time simulation, Multi agent, Mobile robots, Mutual inhibition, Behavior,Pattern recognition, Image processing

ROBOT-IN-THE-LOOP SIMULATION TO SUPPORT MULTI-ROBOT SYSTEM DEVELOPMENT: A DYNAMIC TEAM FORMATION EXAMPLE

by

Ehsan Azarnasab

A Thesis Submitted In Partial Fulfillment Of The Requirements For The Degree Of

MASTER OF SCIENCE

in the College of Arts and Sciences

Georgia State University

2007

ROBOT-IN-THE-LOOP SIMULATION TO SUPPORT MULTI-ROBOT SYSTEM DEVELOPMENT: A DYNAMIC TEAM FORMATION EXAMPLE

by

Ehsan Azarnasab

Major Professor:	Xiaolin Hu
Committee:	Michael Weeks
	Yanqing Zhang

Electronic Version Approved:

Office of Graduate Studies
College of Arts and Sciences
Georgia State University
May 2007

To My Mother Iran.

Acknowledgements

I would like to thank Dr. Xiaolin Hu, my supervisor, for his kind support, new ideas and teaching me new concepts especially in simulation area. I enjoyed being a member of his active research group. I should also thank Professor Bernard Zeigler for inventing DEVS and DEVS Java Package and also letting me use his office to setup the robot environment.

I thank Dr. Weeks for giving really useful hints for writing any text such as this thesis. Also, Professor Zhang gave me new perspectives in AI part of the project. I should also mention that I have used ImageJ, a great free image processing package written in Java, by National Institute of Health.

Of course, I am thankful to my parents for their love and for their constant support in my studies.

Finally, I wish to thank the following: Dr. Sunderraman for his guidance during my studies here; Tammie Dudley (for her kind help); my father to whom I owe my way of thinking, *and* my sister Elham (for she is my only sister).

Atlanta, Georgia Ehsan Azarnasab
December 20, 2006

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Digital computers are *Simulating* the code in the machine. Simulation generally is known as the imitation of the reality. Alan Turing uses the word *simulation* to describe the action of any discrete-state machine when running. These machines have changed our lives because they interact with the outside world. Object oriented design is a successful methodology perhaps partially because it better imitates the reality. As we can see here, from the machine perspective it is hard to draw a distinction between real and virtual objects. Put it in other words, as far as you *behave* like a character in a computer game, you can be substituted in that role. This concept is utilized in Virtual Reality, with many applications for example in flight simulators [1], bio-informatics [2], realtime weather forecasting, business [3] and robotics [4, 5, 6, 7, 8]. The simulation and the models being simulated together are called *modeling and simulation* (M&S). Modeling and simulation can be used as the feedback of the testing phase during system development.

Building large and complex systems often require defining intermediate goals, with conditions that should be satisfied before moving to the successive stages. Simulation can be used for testing the system under development when some parts are missing.

Robot-in-the-loop simulation [9] is a step-wise method which is used in this text for a system of multi robots, while many robots are supposed to collaborate in the final system. The robots each is assigned an ID which specifies its position in a convoy so that a robot with a greater ID should follow the robot with the lower ID. To develop such mechanism, first the robots and environment are modeled and simulated (All virtual robots).

When the transition from simulation to real system is considered, building the system (completely) right after initial simulation is believed to be a big jump. Making too many robots collaborate based on the initial robot (and environment) models for instance is a complicated problem that so many parameters should be considered. On the other hand, in a step-wise simulation-based system development the real robots are added to the convoy in each step so that the real and virtual robots would work together for a better system modeling. Increasing the number of real robots, at some stage the virtual robots are eliminated and the final system would consist of many real robots working together. In the central team formation, a computer is controlling the robots while the results of distributed system (with all real robots running their programs), measures the efficiency of current system design method.

In modeling current system, the robots are homogenous agents interacting based on sensory input (distances to the neighboring agents). The decision making part (of each agent) is based on a network of behavior models, so that behavior tasks define the navigation of robots. The behavior network is modeled in DEVS (Discrete Event System Specification) and can be used as a teaching tool for DEVS and modeling. In this research we apply the method of robot-in-the-loop simulation to a case study example; a dynamic team formation multi robot system. The research is based on the

DEVS modeling and simulation framework. The goal of this project is to implement a robotic team formations (Robot Convoy) in which real and virtual agents interact with each other. Real robots are added at each step to the convoy to analyze the results for a better modeling and design. A Real-time simulation is performed for a robot convoy of up to four Khepra Robots and many virtual robots.

In the first part of this project, a central multi-agent convoy, is developed so that the environment is fully observable and the navigation is managed by the simulator. The next part consists of applying the resulted model to the distributed autonomous robot convoy with and without virtual robots while the environment is partially observable. At each step of this development process, design advances enabled by robot-in-the-loop simulation are presented and performance measurements of the system are given. Finally, the localization of real robots is done by real-time image processing on the specific patterns on top of each robot. The rest of this chapter gives some background on DEVS and behavior network system, then the outline of the remaining text comes.

1.1 Background on Discrete Event System Specification (DEVS)

Modeling and Simulation starts with modeling the real system and builds simulators upon them. The event based nature of the real robotic system makes Discrete Event System Specification (DEVS) a suitable environment with enough power and more efficiency than time based modeling [10, 11]. Availability of the DEVSJava [10] package (written in Java as a portable language with many features including easy concurrency) made DEVSJava the prime choice for starting point. DEVSJava is

used in many simulation problems before, including the simulation of Robot Convoy with behaviors [12], which is the current work too. DEVS is a theoretical modeling which can describe hierarchial and modular systems. Based on the formal definition of DEVS, it is able to model both time based and event based systems. In addition, other modeling techniques which are suitable for the real-time world such as *Petri nets*, *Finite State Machines* and *Timed automata* have DEVS equivalents. The structures of a model may be expressed in a mathematical language called the *formalism*. This formalism defines the way variables take values and the time these values should take effect.

The basic elements in DEVS formalism are *atomic* and *coupled* models [11].

Definition 1.1.1. An atomic DEVS model is defined as

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

where,

X: set of external input events;

S: set of sequential states;

Y: set of outputs;

δ_{int}: $S \longrightarrow S$:internal transition function

δ_{ext}: $Q \times X^b \longrightarrow S$:external transition function

δ_{con}: $Q \times X^b \longrightarrow S$:confluent transition function

X^b is a set of bags over elements in X

λ: $S \longrightarrow Y^b$:output function generating external events at the output

ta: $S \longrightarrow \Re^+_{0,\infty}$:time advance function

$Q = \{(s, e) \,|s \in S, 0 \leq e \leq ta(s)\}$ is the set of total states and e is the elapsed time since the last state transition

To describe the atomic model, suppose the system is in state s. If no external event happens the system will remain in the same state for $ta(s)$ (resting time). If $ta(s)$ is zero then the state would be *transitory* but if $ta(s)$ is infinite the state is *passive*. When the resting time expires ($e = ta(a)$), the system outputs $\lambda(s)$ and changes to the new state $\delta_{int}(s)$ (Fig. 1.1). In this figure, X and Y are incoming and generated events respectively.

In the case an external event ($x \in X^b$) occurs before resting time elapses (system is in total state (s, e) with $e \leq ta(s)$), then the system changes to another new state $\delta_{ext}(s, e, x)$ (Fig. 1.1). The confluent transition function determines the new state if both internal and external transition functions happen at the same time.

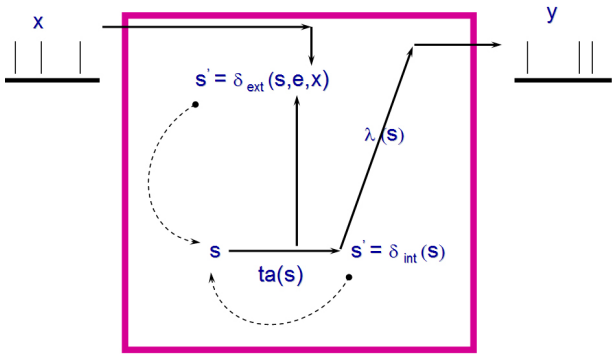

Figure 1.1: DEVS atomic model in action

The basic model contains the following information [10], which are useful to keep in mind when writing the code for basic models:

- the set of input ports through which external events (X) are received.
- the set of output ports through which external events (Y) are sent.

- the set of state variables and parameters: two state variables *phase* and *sigma* are often present, so that the system stays in the current *phase* for the time given by *sigma*.
- the time advance function, value of *sigma* and state variable, if present.
- the internal transition function that specifies to which new state the system will transit after expiration of time advance function.
- the external transition function: based on the port of new input, the value on the port, current state and the time that system has elapsed in current state, external transition function which puts the system in new state.
- the confluent transition function, which determines the new state if an external input and an internal event occur at the same time. One approach in choosing this function would be applying the internal transition function then the external transition function.
- the output function which generates an output before each internal transition function is executed.

This semantics of DEVS decouples inside of the model from outside by the concepts of internal and external events, when both can be present. Figure 1.1 depicts the time elapse concept behind this model. The **input trajectory** (noted by X) is a series of external events occurring at times t_0 and t_2. There might be other internal event times such as t_1 in between with effect on **state trajectory** (noted by S) with step-like series of states changing with each external and internal event. The **elapsed time trajectory** (noted by e) (which has a saw-tooth shape), depicts the passing of

time between events and is reset by each event. Finally **output trajectory** shows the output event produced by output function before applying each internal event.

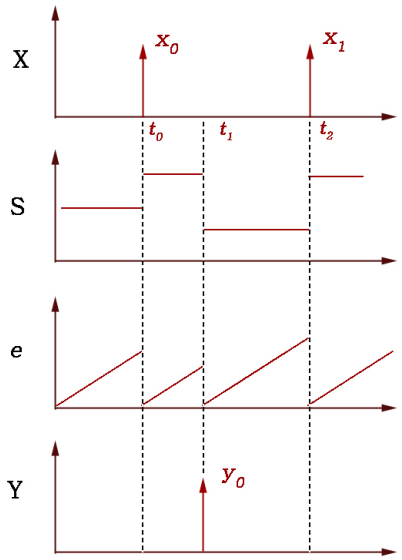

Figure 1.2: DEVS Time Segments

As stated earlier, DEVS models can be coupled in DEVS formalism to build a *coupled model*. Coupled models give a hierarchical structure to the system model in order to form more complicated models.

Definition 1.1.2. A coupled model is defined as

$$DN = \langle X, Y, D, \{M_i\}, \{I_i\}, \{Z_{i,j}\}\rangle$$

where,

X: set of external input events;

Y: a set of outputs;

D: a set of component names' indices;

$\forall i \in D$ [M_i is a component model and I_i is the set of under-influence components for i.]

$\forall j \in I_i$ [$Z_{i,j}$ is the i-to-j output mapping function.]

The coupled model keeps track of all the components, components' influences, the set of input ports receiving external events and output ports sending those events. This model itself has input and output ports connecting to one or more of the components. The formalism is closed under coupling, which means coupled models can be used as a basic model building a hierarchical coupled modeling. Figure 1.1 shows the coupling, in which the entities B,D and E are atomic models and two components A and C are coupled models.

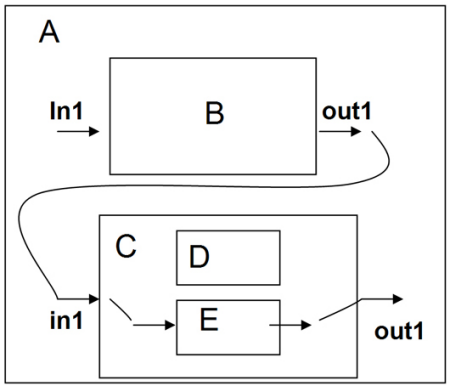

Figure 1.3: DEVS Coupled Model

So far, DEVS modeling does not convey the concept of passing real time, the time which passes with the same pace as the real-world clock. Therefore simulation of such models can be done without synchronization making use of the discrete nature of DEVS for a fast simulation, which is called **"as fast as it can"** simulation (non real-time). This fast simulation is useful in the first step of *model continuity* when no

real part is yet introduced to the system. By bringing real entities to the simulation we need to use real-time simulations. Real-time systems have been a major research area in DEVS [13]. Timeliness requirements of such systems is often the major part in their definition regardless of being embedded or otherwise. Based on definition by Krishna [14] "*a real-time system is one whose logical correctness is based on both the correctness of the outputs and their timeliness.*" By this definition we mean the actual clock, therefore a real-time simulation should have a time synchronous to real clock.

Another characteristic of practical real-time systems is the ability to work with concurrent real objects, having the outputs ready by a deadline and the ability of decision making based on computational processing units. Having that in mind, the real-time simulator objects in DEVSJava are implemented as concurrent *threads.* Therefore the agents in this robotic project are running in separate threads, while the real-time coordinator is responsible for managing the flow of data such as messages passed between ports. However, not all of the real-time requirements can be satisfied without a real-time operating system and a language supporting that. To model real-time systems, based on classic DEVS formalism, real-time DEVS (RT-DEVS) is defined [15].

Definition 1.1.3. An atomic RT-DEVS model is defined as

$$M = \{X, S, Y, \delta_{int}, \delta_{ext}, \delta_{con}, \lambda, ta, A, \psi\}$$

where,

$X, S, Y, \delta_{int}, \delta_{ext}, \delta_{con}, \lambda$: The same as classic atomic DEVS

$ta\colon S \longrightarrow I^{+}_{0,\infty}$: time advance function

where $I^{+}_{0,\infty}$ is the non-negative integers with ∞ adjoint.

A: A set of activities with the constrains.

ψ: $S \longrightarrow A$: an activity mapping function.

Note that unlike classic DEVS, in this definition, the *time advance function* is an integer. This time also is synchronized with real clock time, because the simulation clock is real-time not virtual-time. In DEVS real-time simulation each atomic model is assigned to a real-time simulator that based on the current time decides handling of internal and external events. Therefore unlike conventional object oriented design, a DEVS object provides a mechanism for introducing time in the object [16].

Models discussed in DEVS are implemented in some different languages such as DEVSC++ [17] (in C++) and DEVSJava [10] (in Java). In addition, based on the DEVS concept, the designer just models the system and the simulator is built upon each model automatically. Therefore, one must design the basic models, from which larger models are built, then connect them together (add couplings) in a hierarchical order. These models however, can change their structure and couplings dynamically during the simulation if needed (variable structure DEVS [18]).

1.2 Context dependent Behavior-based modeling and mutual inhibition

Behaviors are often used to model agents when some level of intelligence is required including computer games, crowd behavior [19, 20, 21] or robotics [22, 23, 24, 25]. The behaviors often give the designer the ability to break the agents in simpler parts that together describe a complex system. The interaction between these behaviors can lay upon different architectures like subsumption architecture [22, 23, 24, 25]

in which modules in upper layers inhibit the lower layered behaviors. The mutual inhibition architecture, that we based our modeling on it, is primarily designed for crayfish as a simple animal [26]. This design has shown to be adaptive, robust and with nondeterministic overall behavior for the model.

Animals decide their course of action on a momentary basis in their wild life [27]. Some neural circuits for different behaviors of crayfish are described and the relation of particular behaviors to these circuits is investigated by mutual inhibition [26]. It is important to choose specific set of behaviors, their selection mechanism and the action of each behavior. For example for crayfish, mutually exclusive behaviors such as *forge*, *eat*, *retreat*, *defence*, *escape*, *hide* and *swim* are suggested. The benefit of breaking the model to these behaviors is that when speaking about the animal we often use the natural language for example we say "The animal *eats* when hungry, until the predator is near in that case it may *defend* or *retreat* while finding a place to *hide*".

To model the choice mechanism of behavior in mutual inhibition, each behavior inhibits others according to the level of *activity* weighted by its inhibitory coefficient. The weights model the overall precedence of each behavior. Each behavior in the current design resembles a finite state machine (FSM), that changes its state based on the sensory input. For example, the distance to the food is a parameter that should affect the *eat* behavior's excitation in *BehaviorSim* [27], which is a software for simulating the crayfish based on the behaviors. The behaviors form a network which acts as the decision making unit for the agent being modeled. Since there is no goal defined in mutual inhibition, it is easier to think about the behaviors individually, while the behavior network performs the decision making in an unsupervised manner.

When modeling a simple system by behaviors, considering each behavior like an FSM, the total modeled system would have a predictable outcome (although hard to figure out). For more intelligent agents, it is observed that the behavior pattern may change according to the new situation or context. For example among the crayfish, after forming a social dominance hierarchy the subordinate tends to retreat more than before [12]. This phenomena is often described as the hormonal change among the animals that changes the mood according to the context or motivation. Another interpretion is that the animal has learned from the past, and this new knowledge helps surviving in unpredictable future conditions. To indirectly model this learning mechanism, Hu suggested a two layer architecture [12, 21], in which the top layer is called the *Behavioral Context* layer.

When the simple behaviors provide the simple decisions (like navigational commands), the behavioral context layer provides higher level of control. The top layer performs the structure change by changing the mutual inhibition coefficients. Since these coefficients specify the relative priority of the behaviors, the effect of behavioral context decision would be an immediate change in the total behavioral pattern observed. Based on the discussion above, a possible behavioral context for the crayfish model would have two states of *Dominance hierarchy forming state* and *Stabilized dominance hierarchy state*. For another example, when the modeling of crowd in a building is concerned, two states of *Normal* and *Emergency* may reflect two distinguished moods among the people before and after hearing a fire alarm [21]. The concept of context dependency often leads to more power, for example, a word may have different meanings according to the context.

The general context dependent behavior network architecture is shown in figure 1.2. A behavior (b in behavioral layer) inhibits other behaviors through *inhibitory coefficients*. The coefficient for each pair of behaviors is a real number between 0 and 1, so that a bigger number represents more weight applied for inhibition. Therefore to describe the behavioral layer with n behaviors in a particular time we need an $n \times n$ matrix of mutual inhibition coefficients. Each state (S) in the context layer (top layer in Fig. 1.2), changes the mutual inhibition matrix at a time. Because of the simplicity of action in context layer, it is often modeled as one atomic DEVS model (definition 1.1) with internal states to represent different contexts changed by sensory inputs as external inputs. A behavior ($b = b_i$) reads external input (I_i) and changes its internal state s_i.

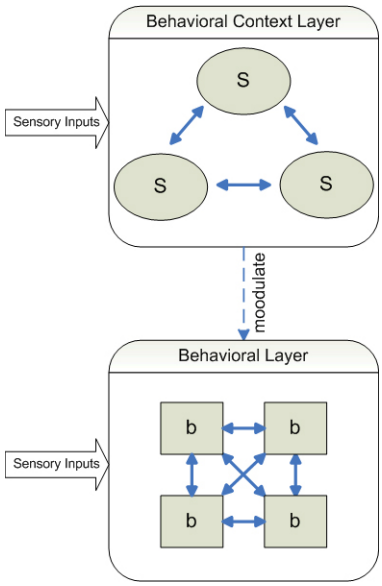

Figure 1.4: Two layer behavior choice mechanism

The mutual inhibition behavior selection describes two mechanisms of excitation

and inhibition of behaviors. A behavior b_i is excited because of s_i and I_i, the excitation magnitude of the behavior is $E_i = E(s_i, I_i)$. While excitation is external, the behavior is inhibited by other behaviors in the same behavior network. The result of excitation and inhibition determines the potential energy of the behavior to take action and is called its *activation level* and is denoted as A. If the activation level of a behavior reaches the threshold of T_{exe} it is called an executable behavior. Since only one behavior is the winner to be executed, if there are more than one executable behaviors, the one with higher activation level (or has waited more to be executed) is chosen to be *active*. The last active behavior remains active if no other behavior is set to be active as the result of the selection mechanism. An active behavior performs its *actions*. There are several ways to perform the behavior actions, one way which is good for *as-fast-as-it-can* simulation would be to use task queues including *resumable* and *non-resumable* tasks. The two kinds of tasks specify if a task should be performed from the point it was left or if it should be restarted from the beginning of the queue. Since the behavior selection is done for a small time scale, it is also possible to perform only one small action at a time when the behavior is activated. Design of behavior actions is further discussed in chapter 4 where based on model continuity the actions are specified level by level.

The second part of mutual inhibition based behavior selection specifies the inhibition mechanism in behavioral layer network. The behavior b_i inhibits b_j based on the inhibitory coefficients ($0 \leq c_{ij} \leq 1$). If, for example $c_{kl} = 0$, then there is no inhibition from behaviors b_k to b_l. If $c_{kl} = c_{lk} = 0$ for all during the simulation we can say that there is no link between these two behaviors in the network, otherwise there is a link between them which means a possible mutual inhibition. If the activation

level of a behavior (say b_i) is greater than or equal to a threshold ($A_i \geq T_{inhibit}$) it can inhibit other neighboring behaviors in the behavioral layer. To calculate the amount of inhibition that behavior b_i imposes on behavior b_j the activation level of the inhibiting behavior (A_i) is multiplied to the corresponding inhibitory coefficient (c_{ij}) from b_i to b_j. The amount of inhibition from b_i to b_j is calculated by equation 1.2.1. After being inhibited by many behaviors, the behavior b_j would have a new activation level A_j, which is calculated by subtracting all applied inhibitions (Equ. 1.2.2).

$$B(i,j) = \begin{cases} c_{ij} \times A_i & \text{if } A_i \geq T_{inhibit} \\ 0 & \text{if } A_i < T_{inhibit} \end{cases} \tag{1.2.1}$$

$$A_j = E(s_j, I_j) - \sum_{i}^{i \neq j} B(i,j) \tag{1.2.2}$$

The model is robust and gives adaptive patterns of behavior. The system starts with an initial inhibitory coefficient matrix, but it may be changed by the context layer or even some learning algorithm. To implement the behavior network one possible design is to model each behavior individually as an atomic DEVS model [27] which requires a selector model responsible for applying the mutual inhibition algorithm as discussed. A more efficient way (which we chose) would be to model the behavior network with one atomic DEVS model keeping the state of all behaviors and updating them. Finally, the agents (as comes in the next chapter) in the multi-robot team formation, each have a behavior network that is the main decision core of that agent.

1.3 Contributions and the outline

The robot team formation system (as a case study for the robot-in-the-loop simulation) is built, this includes both hardware setup and control software implementation

(plus the image processing part). The behavior-based modeling of the robots in the robot convoy (which was implemented before [12], without considering the real robots) is improved step by step using the real-time simulation (in the central team formation) and then applied to the distributed team formation by writing the code for the robots' microchips. The distributed team formation's results will be published in the future papers. Chapter 2 introduces the hardware and most of the software architecture of the project, while the image processing part comes in the chapter 3 because of the more materials it covers. Chapter 4 includes the step-wise simulation procedure of the robot-in-the-loop methodology and improvements of the models. In addition, the idea of robot-in-the-loop is discussed as a general technic to build complex systems.

Chapter 2

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*, 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. Many projects related to robotics involve mathematical solutions to deal with control aspects of a problem, which may result in building a robot and efficiently controlling it. On the other hand, robots can be used as a part of another project, like in our case the model continuity and behavior research. Both problems are in fact engineering problems, involving design and implementation. Tradeoffs should be made to achieve acceptable results, considering the budget and defined goals. Before the beginning of this project the team formation simulation (in *as-fast-as-it-can* mode consisting only virtual robots) was ready [12], this simulation helped in choosing our design goals.

2.1 Robot convoy hardware design

To choose different items to buy and different strategies, the hardware design criteria we chose in this project were:

1. The design should be inline with model continuity.
2. The system should be easy to use.
3. The system should be suitable for teaching purposes.
4. The design should make the hardware issues transparent.
5. The items should be powerful enough to handle their task.
6. the Items which can be used for different purposes are preferable.
7. We should spend resources on the items which are more central to the project.

Figure 2.1: Khepra II robots with mounted serial communication turret

In the current project we used four Khepra® Robots (Fig. 2.1) from K-Team® corporation. These robots are used in many research areas around the world and are shown to be powerful enough for many robotic experiments [28, 4, 29].

In the team formation project, the agents should be able to know at least some information about successive agents they want to follow. One way to achieve this

for real robots would be to establish a wireless link between them, and also between real robots and virtual robots, if there are any virtual robots running there. Wireless communication was through the standard Khepra® serial base. These standard modules also have the ability to be configured as both inter robot communication and transparent serial communication with the central base (normally attached to a PC). We also considered using another cheaper, and more powerful transceivers which are named *RF telemetry module* and are available by Active Robots® which uses EasyRadio®. These modules are more power efficient than standard wireless solution, transmit faster and more reliably and also have a longer range (400 meters compared to 10 meters standard wireless links). We could be able to successfully change and use these modules with one robot, however working with the standard communication is more clear and can be expanded more easily by people. In addition, little involvement of hardware design was more desirable, to make hardware issues transparent and out of the software project which was the goal. The wireless communication with real robots is a part of robot navigation and is implemented as a separate class module, giving more transparency.

From *all virtual* simulation results the need for finding relative distances to some other robots was arisen. The Khepra® II robots have eight infra red sensors which can be used in *localization* and finding distances. However, these sensors are error prone and also this approach for localization would add to the complexity of the control aspect, which was not the major project goal. In addition this approach requires local processing and had to be implemented with programming the robots locally, consuming the small resources of robots (like memory and processing) while we need them for our higher goal (when autonomous robot convoy) which was coding behaviors

inside the robots. In fact, localization of mobile robots based on multisensory inputs (like using 8 infra red sensors on Khepra II) and building a map of the environment is another research area and involves using Kalman Filtering and other data fusion techniques [29, 30].

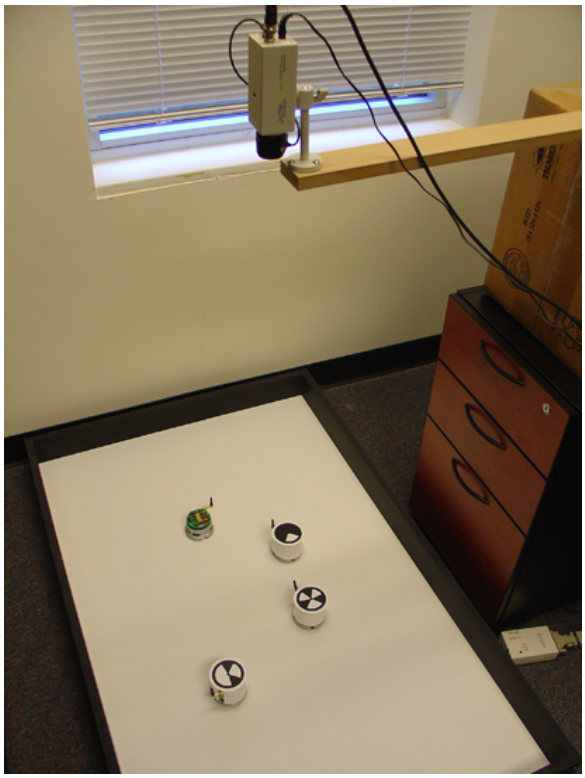

Figure 2.2: System consisting of real robots and real environment

Obviously, that research was not inline with our project design goals. There are several methods for localization of robots such as using laser scanners, sonar sensors, GPS, gyro, vision or a combination of them. Laser scanners are often big, expensive and subject to the restrictions of local processing. The newer version of Khepra® adds ultrasonic sensors to the infra sensors, which have the same usability. Global

Positioning System (GPS), if using available free service by satellites, needs an open space to work but can lead to about one centimeter accuracy in positioning. On the other hand, providing a local GPS is expensive and difficult. Using gyros (or step wheels) for localization involves dead-reckoning thus is error prone. Dead-reckoning is the algorithm of finding the new position based on previous position and other parameters such as the speed. In fact Khepra® robots have a built-in Proportional-Integral-Derivative (PID) controller (definition 2.1.1) which makes dead-reckoning based localization feasible (equation 2.1.1). Although the extensive programming capabilities of Khepra® (in which all parameters of PID controller can be tuned) promises good accuracy, traveling long distances especially on a non uniform platform (which is the real case) makes this method not practical for our purpose.

Definition 2.1.1. A PID controller is a controller with the following transfer function:

$$H(s) = K_P \times \left(\frac{K_D s^2 + s + K_I}{s + C} \right)$$

where,

C: a constant which depends on the bandwidth of the controlled system.

K_P: the proportional gain constant

K_I: the integral gain constant

K_D: the derivative gain constant

Equivalently in time domain we have:

$$Output = P + I + D$$

where,

$$P = K_P.e(t)$$

$$I = K_I. \int_{-\infty}^{t} e(t)dt$$

$$D = K_D.\frac{de(t)}{dt}$$

In discrete time intervals, the PID controller can be written as:

$$Output_{n+1} = Output_n + K_P.e_n + K_D.\,(e_n - e_{n-1}) \tag{2.1.1}$$

The control features of Khepra® makes it an ideal teaching tool for any other future project related to robotics, for example a more comprehensive DEVS model for Khepra® robots including the PID parameters gives better results from agent models because it better approximates the new position of real robots. This can be implemented using available DEVS models for *integral* and *derivation* (Equ. 2.1.1) both available in DEVSJava. Even a better model can be designed by considering more physical characteristics of the platform as a part of the environment.

Finally, after considering different options, we decided to use an overhead camera connected to a Personal Computer (PC) to find the robots in the field. This overhead camera is located so that the grabbed picture covers the entire filed on which the robots move. By this choice we avoided the overhead processing inside each robots and also using available techniques of image processing we could find the location and direction of each robot. 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 (Fig. 2.2).

Because vision might be replaced by another localization technique in the future we decided to use a normal color camera and an ordinary frame grabber. On the other hand, we compensated the low-quality image, with a robust image processing method. Also, the frame grabber we used is *Video Gala* from PixelSmart. This frame grabber can capture color images and comes with a software library. Unlike the part of library written for other languages, the original library did not include a straight

forward method for Java (because of special properties of Java images) , it is managed inside image processing class with using the Java native code.

Furthermore, it is better to place the camera (used for localization) on top of the field in the center and not too high (Fig. 2.2), to minimize the distortion effect of the lenses know as *barrel distortion* and avoid further overhead of correcting this error. As we see in the next chapter, the image processing is done on gray scale images and technically we could use black and white camera and frame grabber, but having the color picture in hand might be useful for future project expansion (for example to a three dimensional environment). In addition, a colorful image is better for demonstration and teaching purposes.

The choice of image processing for localization suggests using a plain white-surface platform where the robots are placed. Figure 2.2 shows the *system* consisting a rectangular platform (field) with robots moving on the field and an overhead camera on top of it. The wireless base is also visible at the bottom right of the picture. In this picture, only three robots have a special pattern which is used for localization.

2.2 Robot convoy software architecture

The main focus of the project is the software engineering aspect. Software is like a controller responsible for managing data flow and interacting with the outside world. Software should be able to get some experimental parameters from the user, control the robot convoy system and provide a monitoring system to display the current state of the system to the user. Based on this criteria, the main components of the system are devised (Fig. 2.3). In this design, the user can interact only with the *control panel* through its Graphical User Interface (GUI in Fig. 2.4), therefore the internal

architecture of the application is transparent from user point of view.

When the software is run, control panel takes control of other components. 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* (upper tabs in Fig. 2.4) and start or stop the real-time simulation of the system. Vision and communication can be present in manual navigation or in the realtime simulation modes named above, therefore they are implemented in another part of the GUI (right hand side tabs in Fig. 2.4). If vision is decided to be the localization means, user can start vision and monitor the video taken from the platform, later the realtime simulation adds virtual data to the video for more clarity of the system. These interactions are shown in figure Fig. 2.3 by arrows starting from the initiators.

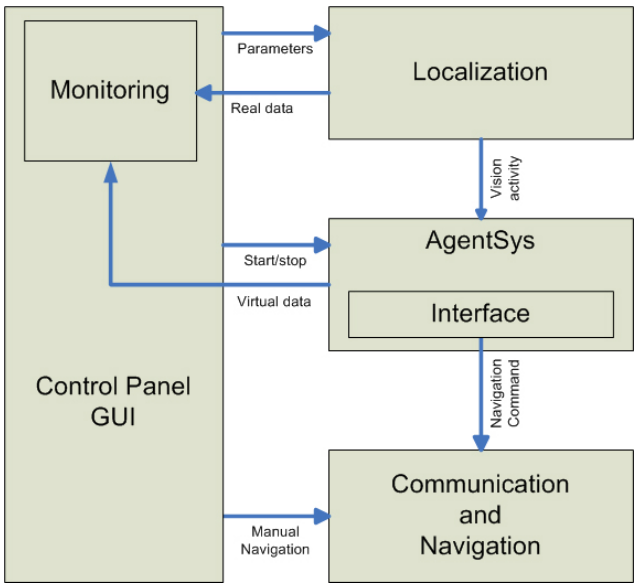

Figure 2.3: System architecture

Based on the functionality of each component and overall performance, object design characteristics can be decided. DEVS modeling can be adopted as the entire software engineering problem, but an efficient choice would be to use the pure object-oriented for the parts which do not involve directly in system modeling process. Therefore, control panel, vision and navigation units are treated as objects.

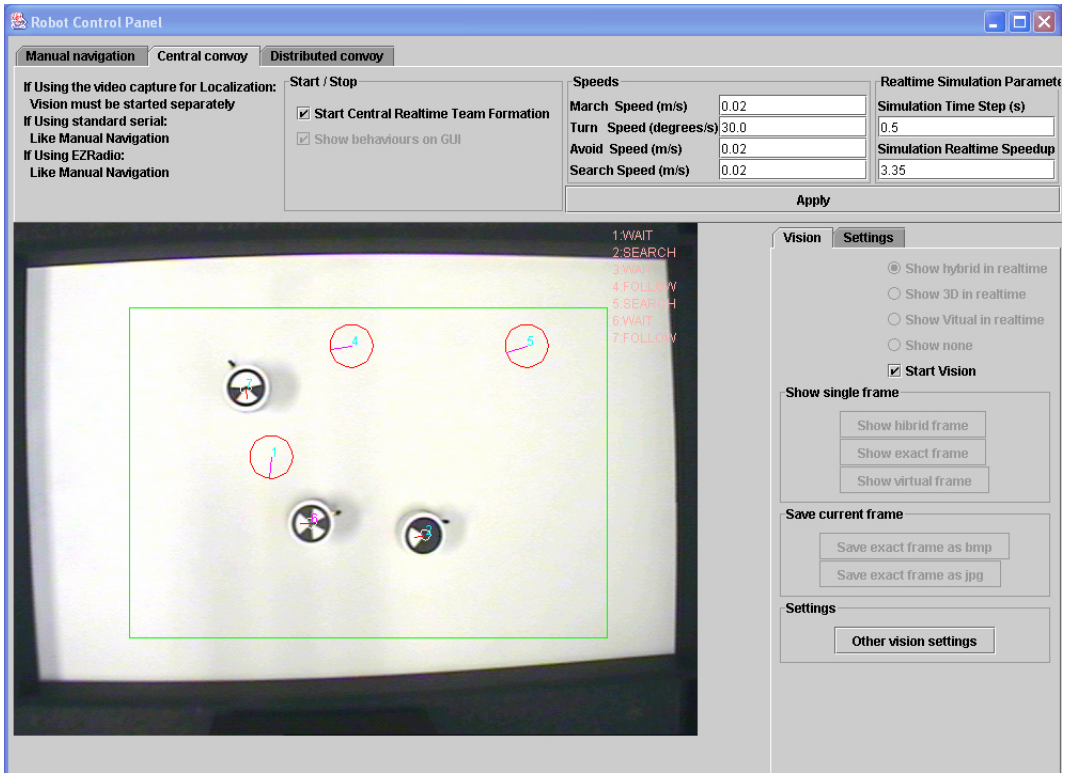

Figure 2.4: The control panel GUI

On the other hand, robot convoy control is based on the modeling of the whole system. Control actions are taken by real-time simulation as the central component of the software design. In order to achieve the model continuity design and implementation pattern, the software main DEVS model is basically a model for the real

system shown in the figure 2.2. This main model consists of the environment and the agents and is called Agents System (*AgentSys*) even inside the software. Figure 2.3 shows AgentSys interaction with other components.

Finally, as a design goal the interactive GUI was designed (Fig 2.4) to enhance the application. The user can start or stop real-time simulation, see the virtual and real robots together, watch current behavior of agents, and also see the effect of changing some parameters like robot speed or boundaries of the environment. This part should be combined with the result of another project (now in progress) in which all behaviors (as well as the behavior network and mutual inhabitation parameters) are defined graphically. Then we would have a complete platform for testing the concepts of DEVS modeling, simulation and behaviors. However, currently the behaviors are hard coded.

2.3 Agents System

When the real-time robot convoy simulation starts, the real-time coordinator (main simulator inside *AgentSys*) takes control of all real and virtual robots together. While the localization unit supplies the real data (like the video taken from the field) to the display GUI, AgentSys adds virtual data (like virtual robots and boundaries of virtual environment) and current behaviors of all agents to the same GUI as a virtual layer on top of the video (the right upper part of Fig. 2.4). Agents system consists of the DEVS models for the real system (Fig. 2.2) including environment and agents. Figure 2.5 illustrates this design.

The environment is a real-time atomic DEVS model for everything but the robots in the system (Fig. 2.2). So logically it should be able to access all data and events

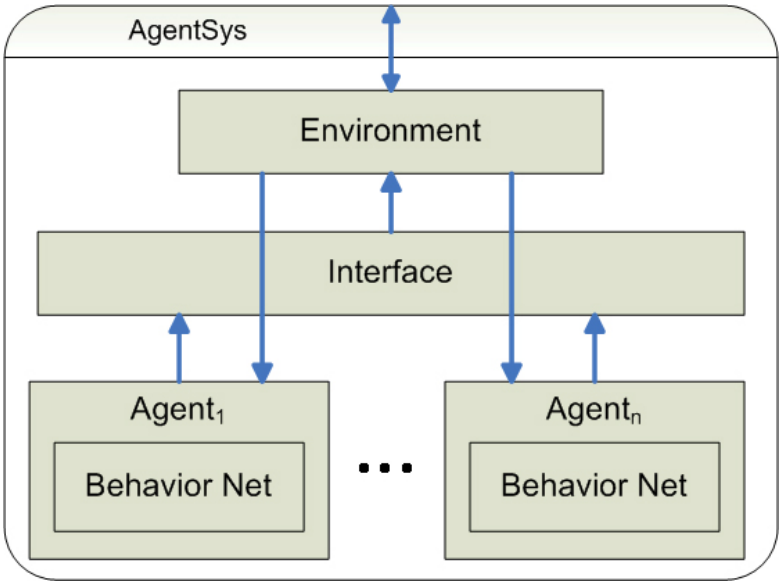

Figure 2.5: Agents-system model

of the system. 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. 2.4), rectangular boundary of the virtual environment (green rectangle in Fig. 2.4) 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 (definition 1.1.3) and closes the loop of the control system. 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 (Fig. 1.2) as the decision making part. Robot

team formation (in 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 possibly changes the position or heading of the virtual robot. In another words, if no noise is added to the navigational commands of the virtual robots, they behave like perfect real robots.

2.4 Localization

The answer to the question

Where am I?

is given by *Localization*.

Localization can be done on-board which means each robot finds its location by some means, or an outside system can locate each robot and possibly inform them. The latter acts as an onlooker outside the system and it is often easier to implement especially for multi agent systems. Different methods for localization of robots exist and it has always been an active research area in robotics and avionics [30, 29]. As

stated at the beginning of this chapter we use image processing for localization of real robots (section 3.4). Obviously, the virtual robots do not need to be localized since the computer already knows their position and direction.

As the diagrams in figures 2.5 and 2.3 suggest, the result of real-time localization of real robots should update the environment. As environment is a real-time atomic model(refer to the definition 1.1.3), it runs a DEVS activity (called vision activity) to get the new position and heading of all real robots. This design is more efficient than running a separate activity by each real agent model, especially based on our design there is no distinction between real and virtual agents but in the environment interface. Note that there is no localization for virtual robots and their position is updated by agent model inside navigation. The vision technique as localization is further discussed in the next chapter.

2.5 Navigation

The output of a controller should finally be converted to navigational commands to move the robots in certain way satisfying some conditions (Fig. 2.7). A controller for a robot is often designed to ensure criteria such as having desired speed, or position or energy consumption. Different robots may have a variety of navigational commands, a humanoid robot may have high level commands for robot to walk.

Khepra® is a mobile robot with two side wheels (Fig. 2.1 and Fig. 2.6). By controlling the speed of these wheels robot can be navigated to different directions and with different speeds. Equation 2.5.2 shows how both displacement velocity v and angular velocity ω can be calculated. Where V_R and V_L are the speed of right and left wheels and D is the diameter of the circular robot (Fig 2.6).

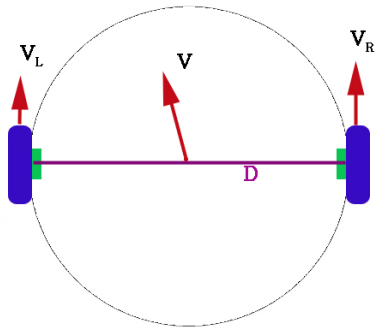

Figure 2.6: Differential Drive; V is the speed vector of robot determined by the speed of the left wheel (V_L), right wheel speed (V_R) and robot diameter (D)

$$\begin{aligned} V &= \tfrac{V_R+V_L}{2} \\ \omega &= \tfrac{V_R-V_L}{D} \end{aligned} \tag{2.5.2}$$

Based on equation 2.5.2, for a robot to move forward with speed v_0 both wheels should have the same positive speed (or $V_L = V_R = v_0$). The robot will move backward if the speeds are the same but in the opposite direction (or $V_L = V_R = -v_0$). By changing the sign of V_L and V_R, but keeping them with the same magnitude, robots rotate clockwise (CW) or counter clockwise (CCW) around the center of the circle. This formula does not consider the effect of a non-uniform platform or difference of fractional characteristics of the wheels. As a result although the virtual robots move the same when receiving the same navigational command, it is not the case for real robots because of the intrinsic heterogeneously of them.

For example, if a virtual robots receives the navigational command of 1 radian clockwise rotation, the virtual environment interface increases the heading of that

agent for 1 radian. Receiving the same command, by an agent corresponding to a real robot, the real environment interface calculates the speed of each wheel (V_L and V_R) (based on Equ. 2.5.2 and desired rotation speed given in the control panel) and transmits them to the real robots using the communication link.

2.6 Communication link

Communication among agents and the the simulator is performed by the message passing mechanism of DEVS. Each DEVS entity can have some ports to send and receive data. When integrating the virtual and real parts, there should be a means to really transmit these messages for real entities, we call this part the *communication link*. Communication link can be wired or wireless, for example Khepra® robots can receive commands using a cable connected to the serial port of the computer. The advantage of wired link is the perfect data transition, eliminating the need to retransmit packets, thus more possible transmission rate. On the other hand by explicit usage of wired link for more than one robot, many serial ports on the computer side are needed, also the presence of many cables hinders the free movement of the robots.

Based on extensibility design goal, we preferred using wireless links from the early part of the project. The standard wireless communication of Khepra® comprises on-robot wireless turrets (shown in Fig. 2.1 with the antenna) and one base transmitter (bottom right in Fig. 2.2). The commands are sent to the base by a serial cable from the computer, and the base dispatchs them according to the receiver robot. Figure 2.7 shows the navigation and communication unit. The navigation unit is responsible for translating the navigational commands by interface from AgentSys. The standard

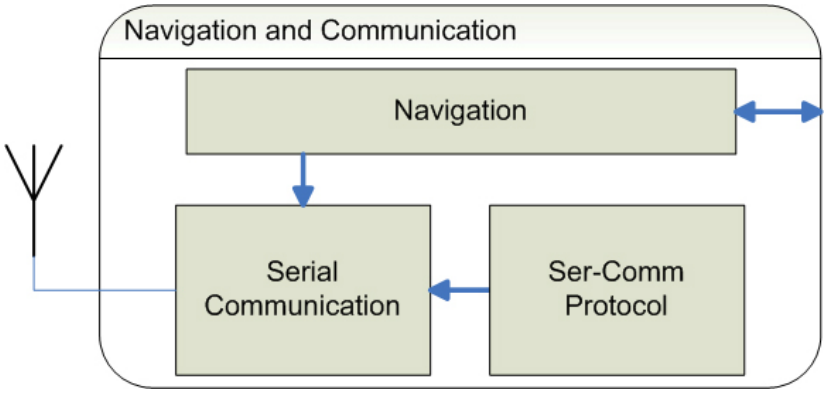

Figure 2.7: Navigation and communication unit; The serial communication in this figure is connected to an antenna (outside the block), representing the wireless communication

Khepra® serial communication protocol is called *SerComm*, this protocol specifies the format of the commands as well as some standard navigational commands. The final navigational commands are based on SerComm [31, 32] and are transmitted through a serial communication link to the base then by antenna. By sticking to these standards, the available on-board software of the robots can be used for the central robot convoy.

In the central team formation the navigational commands are sent over the link, while in the distributed team formation the positions and headings of the robots are transmitted to the on-board software so that each robot can decide its own navigation based on the embedded behavior network. In the central team formation the navigational commands should be sent to the specific robots. However, in the distributed team formation the locations (found by image processing) can be broadcasted to all robots, making use of the available bandwidth. In addition, the implemented on-board software of the robots does not echo back all the commands (unlike the

SerComm protocol), which leads to more channel utilization. Table 2.1 shows the protocol for some of the commands which are written for the on-board software. Each command starts with one letter indicating the its function and then the command parameters are followed. At the end of each command a carriage return (CR) or line feed (LF) follows, to indicate the end of the command. The parameters are delimited by commas. For example, the command of "*D,1,-1*" sets the speed of the left and right wheels to $\frac{1}{127}$ and $\frac{-1}{128}$ meters per second respectively, in which the letter D is the displacement command with a similar effect in the standard SerComm protocol. To be consistent with the standard protocol, the unused commands in SerComm [32] are expanded to support the new needed commands. As a result, the developed on-board software can be used for the central team formation as well.

Command	Explanation
X,x1,y1,x2,y2,...[CR or LF]	Sets the (x, y) locations of the robots in the field (in meters).
Y,h1,h2,...[CR or LF]	Sets the headings of the robots (in radians).
D,s1,s2[CR or LF]	Sets the speed for the left and right wheels respectively (in meter per second).
S,c[CR or LF]	If $c = 123$ the distributed team formation starts, otherwise it stops. (Command has an echo)

Table 2.1: The communication protocol (incomplete), each command starts with a letter (indicating its function) ,then a list of comma delimited parameters and finally a carriage return (CR) or line feed (LF) character.

Chapter 3

Vision and Image Processing

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 [33]. To find robots more accurately, while not relying on robot processing abilities, we can use an over-head camera to track the movement of robots [34]. For simplicity of design, the latter method was chosen, so the problem changed to finding particular *objects* (here robots) in the image taken from a field by overhead camera. Current team formation requires each robot to follow another particular robot, therefore we should associate each of the robot objects with a number. In addition, the mechanical structure of Khepra® (with two wheel drive) requires direction for each robot for navigation. To solve the new problem, we needed to put different *markers* (Fig. 3.1) on top of each robot (three robots in Fig. 2.2). These markers should be

informative enough to distinguish at least 4 robots (in current phase of the project) from each other with accurate location and heading.

Another problem in image processing is called *barrel distortion*, which is the non-linear effect of lenses to the image taken by camera (can be seen Fig. 3.13 where the platform is not a perfect rectangle). This phenomenon is more visible for larger objects in the image. One way to correct the image is to use non-linear transforms used in rubber-fitting methods to restore the original shapes in the picture having some known mapping of the pixel data and objects. A suggested method is to use a grid-image on the field and mapping the grid to perfect rectangle which is desired. Although it adds to the processing overhead, implementing this technique should be one of the future tasks of the project. However, the robust method of image processing could solve this image distortion problem especially for small scale objects. Furthermore, because of the non-uniform nature of barrel-distortion when choosing the pattern for markers these patterns should be robust enough. Also, when analyzing the patterns (in order to get information), a shorter path seems better because it corresponds to smaller objects, hence less error. Periodic nature of circular patterns was one of the solutions with good robustness towards non-linear distortions. Taking into account the circular shape of current robots we decided to use uniform circular patterns for markers.

3.1 Six angle circular patterns

To identify a pattern on a marker, these patterns should be binary coded using the shapes and colors (each code is named an ID). We can consider the image taken from these patterns as digitized patterns. There is some error involving digitization which

is intrinsic to the nature of sampling. Also digitized patterns often contain noise which comes from the poor quality of the camera or frame grabber. Different methods exist to classify the taken images. To speed up the image processing techniques such as *Independent Component Analysis* (ICA), *Principal Component Analysis* (PCA) and recently *Optimal Component Analysis* (OCA) [35] are known to use the pixel data to reduce dimension in an efficient way. However most of these techniques are time consuming because they consider all of the pixel level data mainly because in general the images to be classified are considered to be of any pattern. On the other hand the periodic nature and uniformity of data towards one dimension in uniform circular patterns make the two dimensional marker-image like one dimensional signals. to put it in another words, although the whole image taken from the markers is a sampling, we can perform sampling once more (along the non-uniform dimension). Figure 3.1(a) shows the designed 6 angle pattern. This patterns consists of 6 pie slices, one inner hole and two outer rings. To color code the pattern we use the *pattern template* scheme in Figure 3.1(b), 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.

Since in the current stage of the project we have only 4 real robots this pattern can identify all of the robots. The biggest pie slice (always white) together with its two neighboring black slices can be used to find the direction of robots (heading). Because the platform in the system (Fig. 2.2) has a white surface and black walls, by choosing the outer ring to be white and inner ring to be black, markers (which cover the robots in top view of overhead camera) can be identified well when robots

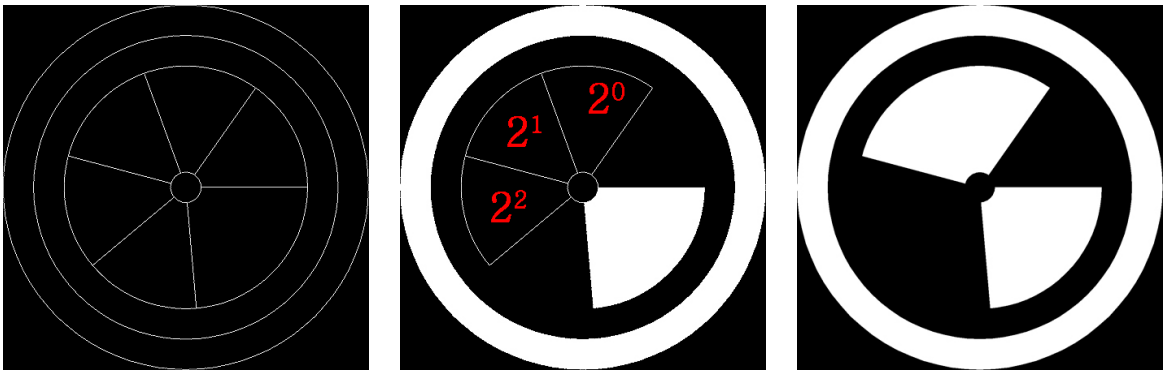

Figure 3.1: From left to right, a) The general six angular pattern b) Pattern template for binary coding c) Pattern for the ID equal to $4{=}1 + 2^0 + 2^1$

are inside the field and even when they go near the wall. Refer to pattern analysis (section 3.2), the inner hole is decided to be black to get a sharper image near the center (which is near the *sampling ring*) this choice also helps in better *blob detection* (section 3.4).

3.2 Pattern analysis

We can define a uniform circular pattern having the number of pie slices in the pattern together with the magnitude of each angle of associated pie slice, and the binary coding scheme.

Definition 3.2.1. A 6-angle-uniform-circular-pattern (6UCP) is defined by three vectors and one coding formula:

$$P_6 = \{\theta = [\theta_1, \theta_2, \cdots, \theta_6], \beta = [\beta_1, \beta_2, \cdots, \beta_6], \alpha = [\alpha_1, \alpha_2, \cdots, \alpha_6], \phi, r_1, r_2\}$$

where,

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

$$\sum_{1}^{6} \theta_i = 360^\circ \tag{3.2.1}$$

$\beta_i \in \{0,1\}$ for $i = 1,2,3,4,5,6$. β is the color vector so that if β_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:

$$\phi = \sum_{1}^{6} \beta_i \times 2^{\alpha_i} \tag{3.2.2}$$

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 3.2.1 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. 3.1) is as follows:

$$\begin{aligned} \theta &= [85,55,55,55,55,55] = 55 \times [1.5,1,1,1,1,1] \\ \beta &= [1,0,x_1,x_2,x_3,0] \\ \alpha &= [0,0,1,2,3,0] \end{aligned} \tag{3.2.3}$$

Note that x_1,x_2 and x_3 refer to the three labeled pies in Fig. 3.1, 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* (CXCov) can detect this pattern because it minimizes the phase. The very first angle is chosen to be 1.5 times other angles, to distinguish this pie as beginning, and maximize the penalty if it is misplaced as another pie when matching patterns. However, a careful choice of the angles can lead to better results in practice.

As stated earlier the ID is coded on the color in angular manner. Assume the circle containing the pattern is known (for example by blob detection in section 3.4), therefore we have the center of the pattern. Now we take our samples counter clockwise on a circle to build the sampled signal. The circle at which we build our sampled signal is called *sampling ring*. The ideal signal built in this way will consist of 6 horizontal slots (on horizontal axis) each with a length proportional to elements of θ vector, the sampled value (on vertical axis) will be 0 or 1 depending on the elements of β. Because of pattern uniformity, ideally there should be no difference at which radius we start sampling, however as mentioned earlier because of the barrel distortion a smaller r_s is better ($r_1 < r_s < r_2$). On the other hand if r_s is too small the number of samples do not suffice to build a near ideal signal, thus the radius sampling ring should be tuned. This parameter is one of the factors that can be modified as vision settings (Fig. 3.12). Also we should consider that we compare the sampled signals to the ideal signals, each time the sampling radius changes a new ideal signal for all of the 8 possible IDs is built. By increasing r_s ideal and sampled signals expand in time with the same proportion.

We are sampling at the ring with radius equal to r_s from a circle previously sampled as an image (2D sampled). To cover all pixels of the circle with perimeter

equal to $2\pi r_s$ we need to sample with interval of δt:

$$\delta t = \frac{1}{2r_s} \tag{3.2.4}$$

As a result the total number of pixels sampled would be $4\pi r_s$. It is easy to show that the length of the signal corresponding to n degrees would have the total of $\frac{2n\pi r_s}{180}$ pixels. For instance, in the ideal signals based on conventional notation (Equ. 3.2.3), the first segment (of total 6 segments) of the signal has the length $\frac{2(85)\pi r_s}{180}$. Figure 3.2 depicts the ideal signals normalized in horizontal axis between 0 and 360 degrees. The first segment, which has the length 85, corresponds to θ_1 and is 1 because $\beta_1 = 1$. All other 5 segments are 55 in length. Also the second and sixth segments have magnitude of 1 ($\beta_2 = 0$ and $\beta_6 = 0$).

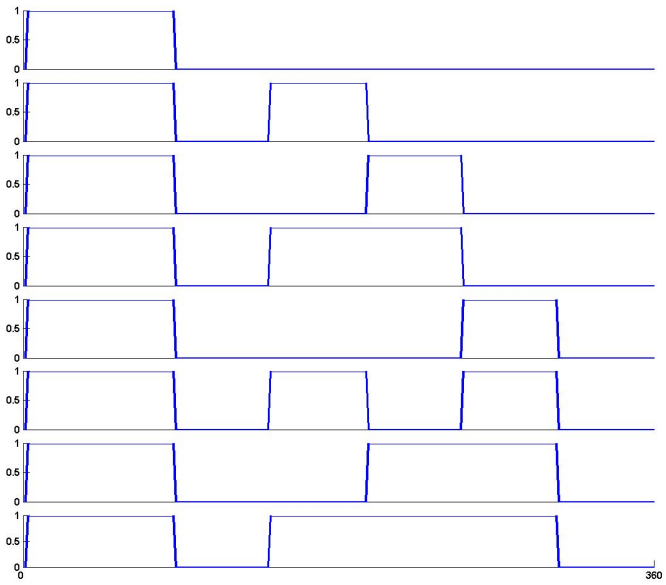

Figure 3.2: Ideal signals from top to bottom corresponding to IDs from 1 to 8

Sampling starts with 0 degrees (in polar coordinate) 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) (Fig. 3.3(a)). In addition, there is no noise in the ideal signal.

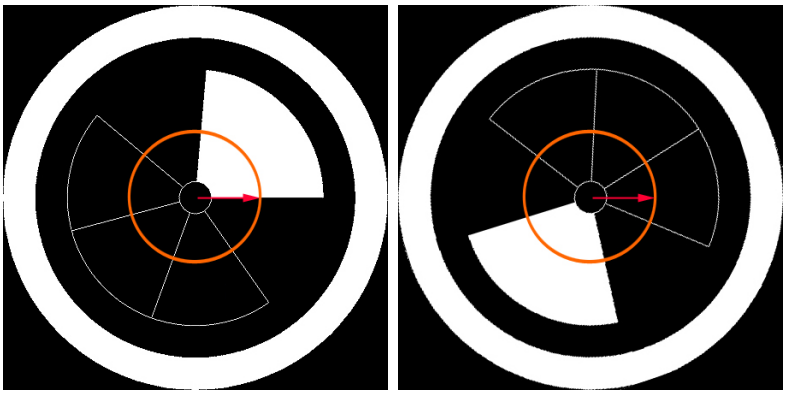

Figure 3.3: From left to right, a) Building ideal signal (no rotation) b) Sampling possibly rotated patterns

However, the pattern might be rotated (Fig 3.3(b)) 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 (Fig. 3.12). Figure 3.4 shows the effect of rotation and noise, as we can see in horizontal axes of this figure the ideal signal is between 0 and 1 but the sampled signal has a range of about 250. Both horizontal axes are normalized between 0 and 360 degrees.

Fig. 3.4 shows the ideal and sampled signal for ID equal to six. The codomain range of the sampled signal (Fig. 3.4(b)) corresponds to the maximum amount that one byte can hold. Although this one byte could be the span of gray scale image, we used the red-band of RGB image with almost similar result. There are two buttons in vision setting GUI with which both ideal and sampled signals can be viewed

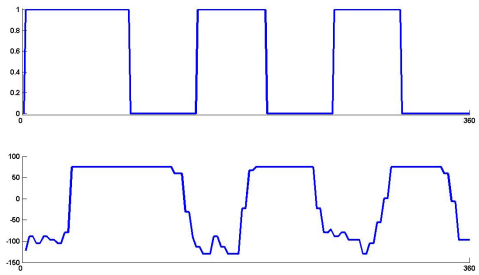

Figure 3.4: From top to bottom, a) The ideal signal for ID = 6 b) Sampled signal for ID = 6 (Both signals are normalized horizontally between 0 and 360 degrees.)

(Fig. 3.12), this can help choosing efficient parameters such as brightness, saturation, contrast, hue and also the average red-band value for white and black colors (all tunable in the same GUI in the middle). In addition, the user can zoom in and out of the image taken from the pattern (Fig. 3.12 top-left), and by moving the mouse cursor on this image, the user can see the values for red, green and blue bands.

So far we have built ideal signals for 8 possible patterns to be identified (Fig. 3.2) and saved them inside a bank, in the next stage by sampling the real marker around the circle we made the test signal (Fig. 3.4 (b)). 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. For example by correlating the sampled signal in (Fig. 3.4 (b)) with signals in (Fig. 3.2) we should come to (Fig. 3.4 (a)). The comparison method should eliminate the effect of different gains and rotation, also it should be robust enough against the noise. To satisfy these conditions we used the *Circular Cross Covariance* (CXCov) criterion and applied it for each two pairs of sampled (or test) signal and signals in the bank. The circular cross covariance is the normalized circular cross correlation function of two vectors (x and y) with their

means removed:

$$CX(d) = \frac{\sum_{i=1}^{N} \left[(x(i) - m_x) \left(y(i-d) - m_y \right) \right]}{\sqrt{\sum_{i=1}^{N} (x(i) - m_x)^2} \times \sqrt{\sum_{i=1}^{N} (y(i) - m_y)^2}} \tag{3.2.5}$$

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

$$y(-k) = y(N-k) \tag{3.2.6}$$

Because d in this formula is the applied lag of one signal, by maximizing correlation ($CX(d)$) over d, we get the signal with minimum shifting (Equ. 3.2.8). Also, because of the normalization part (division by the norms), the gain of the sampling and also the gain of θ vector (55 in Equ. 3.2.3) are eliminated. Equation 3.2.6 emphasizes on the circularity of $CX(d)$. It can be proved that equation 3.2.5 gets its maximum value (which is 1) when two identical signals (x and y) match the same. Therefore we expand this criteria to be used to compare similarity of two possibly non identical signals.

$$\eta(x, y) = max_d \left(CX(d) \right) \tag{3.2.7}$$

It can be proved that similarity (η) is a real number always between 0 and 1. Furthermore, back to the Figure 3.4 and Figure 3.3, the amount of shifting we need to get maximum similarity corresponds to the rotation of the pattern.

$$\lambda(x, y) = argmax_d \left(CX(d) \right) \text{ is the } d \text{ which maximizes the } CX(d) \tag{3.2.8}$$

By applying equation 3.2.4 the rotation can be calculated as:

$$r(x, y) = \frac{360\lambda}{4\pi r_s} = \frac{90\lambda}{\pi r_s} \tag{3.2.9}$$

The rotation (r) is calculated in degrees, and points to the zero degree in polar coordinate of non-rotated marker (red arrow in Fig. 3.3 (a) pointing to the beginning of the largest pie slice). By definition the heading of the pattern (thus the heading of the robot) is defined as the arrow heading to the middle of the largest pie slice(β_1 in Equ. 3.2.3) therefore heading of pattern p with ideal signal of y and sampled signal x is :

Definition 3.2.2. $h(p) = r\,(x, y) + \frac{\beta_1}{2}$

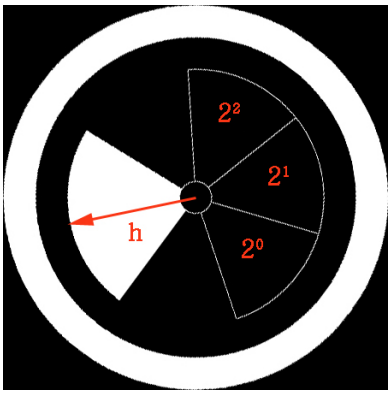

Figure 3.5: The template pattern including the heading

Figure 3.5 shows the heading for the particular instance of 6UCP in equation 3.2.3. Also, Figure 3.13 shows three robots and the heading of which all pointing towards the middle of pie slice of β_1 degrees, the heading is illustrated by a red line beginning the center. It can be seen that when computing η, both r and h are automatically calculated, therefore we can focus on the equation 3.2.7. In general there might be more than one pattern on the field (for example n) for all of which η should be calculated for all 8 possible signals in the bank, resulting in $8n$ different η (for $8n$ comparisons). Suppose there would be no noise involved, the result would be

a table of similarity between ideal signals mutually (Table 3.1).

ID	1	2	3	4	5	6	7	8
1	1.0000	0.7759	0.7799	0.6587	0.7799	0.6587	0.6611	0.5823
2	0.7759	1.0000	0.6051	0.8489	0.7898	0.8489	0.8346	0.7505
3	0.7799	0.6051	1.0000	0.8445	0.6186	0.5224	0.8477	0.7466
4	0.6587	0.8489	0.8445	1.0000	0.8445	0.7206	0.8709	0.8841
5	0.7799	0.7898	0.6186	0.8445	1.0000	0.8445	0.8477	0.7466
6	0.6587	0.8489	0.5224	0.7206	0.8445	1.0000	0.7159	0.8841
7	0.6611	0.8346	0.8477	0.8709	0.8477	0.7159	1.0000	0.8808
8	0.5823	0.7505	0.7466	0.8841	0.7466	0.8841	0.8808	1.0000

Table 3.1: Mutual circular cross covariance of 6 angular patterns

Each cell (in row i and column j) in this table is the η between ideal signals of 6UCP with IDs i and j respectively. To put it in other words, it shows the similarity between these patterns. As can be seen in this table, the main diagonal of the matrix is 1 which means that each signal is 100% similar to itself. Another observation is that the similarity between two non-similar signals is not zero. As a result this table gives a criterion to find the distinctness of two patterns. For example, the distinctness between IDs 8 and 1 is more than distinctness between IDs 8 and 7. Therefore if there are only two robots and three markers (with IDs 1,7 and 8,) logically we choose the IDs 8 for the first robot and 1 for the other. Basically we are trying to maximize the distinctness. Another approach to deal with numbers is to look at them as a Fuzzy relation with universe U and membership function μ_F.

Definition 3.2.3. The Fuzzy relation corresponding to the pattern is:

$$F = \{((i,j), \mu_F(i,j)) \,|(i,j) \in U\}$$

where,

U=$\{1,2,3,4,5,6,7,8\} \times \{1,2,3,4,5,6,7,8\}$ is the universe

$\mu_F\,(i,j)$=$\eta\,(S_{ideal}(i), S_{ideal}(j))$ is the membership function.
$S_{ideal}(k)$ is is the ideal signal with ID equal to k
η is the similarity function defined in equation 3.2.7

So the universe is the relation of 1 to 8 with itself, and the membership function would be the similarity of ideal signals. Figure 3.6 shows the membership function (the same as Table 3.1). This figure helps more when choosing distinct patterns, because we simply should choose those cells which are darker.

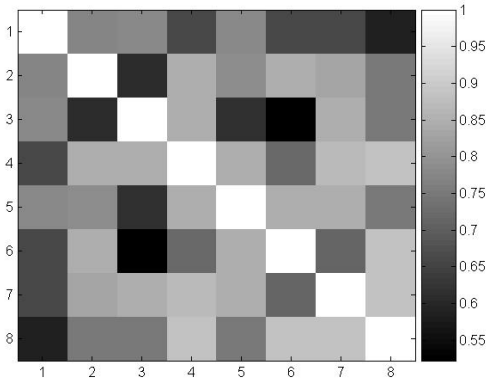

Figure 3.6: Similarity membership function of 6UCP

The final step in analyzing 6UCP is to see the effect of noise (unwanted error). One part of the error might be because of the barrel distortion but as we discussed by choosing an appropriate sample ring it can be to some extent eliminated. Furthermore so far we assumed the center of the pattern is known and is exactly the same as the measured center. Refer to section 3.4, the center of patterns are found in blob detection part (using the centroid measurement). This method is to some extent robust towards the unwanted transformation of the image because usually the transformation is near linear for one marker (Fig. 3.13 shows that the markers are

small compare to the field). The other factor which has a major contribution to the error is error in estimating the center. Figure 3.7 depicts the effect of sampling with erroneous center for a particular pattern. To find out the effect of erroneous center we first formulate the sampling around the circle then we see the effect of the center shifting. The error (shown as a vector e in Fig. 3.7) specifies the shift from the true center to the wrong center. Error (e) can be written in the polar coordinate system as:

$$e = \sqrt{e_x^2 + e_y^2} \tag{3.2.10}$$

Where e_x and e_y are two horizontal and vertical components of error.

$$e_x = e \times cos(\varepsilon) \tag{3.2.11}$$

$$e_y = e \times sin(\varepsilon) \tag{3.2.12}$$

Where, e is the magnitude of the error and ε is the direction of the error. Note that in practice $e < r_1$ (r_1 is the inner hole radius defined in definition 3.2.1).

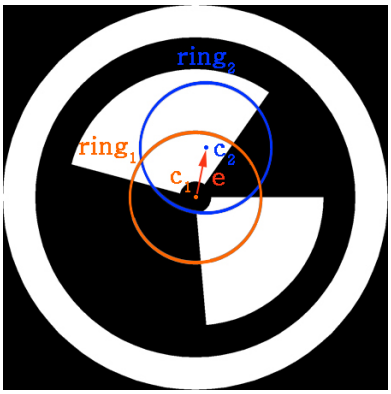

Figure 3.7: Sampling on wrong ring ($ring_2$) instead of $ring_1$

To do sampling we should break the signal into units. The unit function is a step like function.

Definition 3.2.4. The unit function is:

$$u_a(t) = \frac{sgn(t-a)+1}{2}$$

where,

sgn is the sign function.

The output of unit function $u_a(t)$ will be 1 if t is greater than a otherwise it returns 0 (Fig 3.8(a)). One natural result of this function is that if $b > a$ then $u_a(t) - u_b(t)$ is 1 between a and b, otherwise it is zero.(Fig 3.8(b))

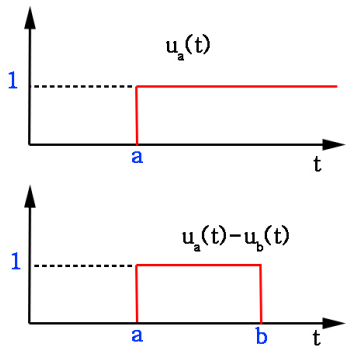

Figure 3.8: From top to bottom a)unit function $u_a(t)$ b)$u_a(t) - u_b(t)$

By eliminating the need to conditional functions, the difference $(u_a(t) - u_b(t))$ can be used to build our sampling formula.

$$B(r,t) = (U_{r_1}(r) - U_{r_2}(r))\,[B_1(t), B_2(t), B_3(t), B_4(t), B_5(t), B_6(t)] \tag{3.2.13}$$

$$B_i(t) = \beta_i \left(U_{\sum_{j=1}^{i-1}\theta_j}(t) - U_{\sum_{j=1}^{i}\theta_j}(t) \right) \text{ where } i \in \{1, 2, \cdots, 6\} \tag{3.2.14}$$

Where r_1,r_2, θ_j and β_i are defined in definition 3.2.1 and B is the sampling vector. One result from equation 3.2.13 is:

$$S_{ideal}(k,t) = B(r_s,t) \tag{3.2.15}$$

Where $S_{ideal}(k,t)$ (also defined in definition 3.2.3) the is k^{th} ideal signal sampled at the angle equal to t. (Also consider that pattern parameters such as β and θ depend to k). To find the sampled signal in presence of error $S_{sampled}$ we consider the shifting of center (Fig. 3.7) in the polar coordinate system.

$$S_{sampled}(k,t) = B\left(\sqrt{r_x^2\left(e_x,t\right)+r_y\left(e_y,t\right)^2}, atn_2\left(r_y\left(e_y,t\right), r_x\left(e_x,t\right)\right)\right) \tag{3.2.16}$$

Where, $atn_2(y,x)$ returns the angle (between 0 and 2π) of a vector with components x and y. The transformed radius components r_x and r_y can be calculated by:

$$r_x(e_x,t) = e_x + r_s \times cos(t) \tag{3.2.17}$$

$$r_y(e_y,t) = e_y + r_s \times sin(t) \tag{3.2.18}$$

Where, e_x and e_y are horizontal and vertical components of the error vector (Equations 3.2.11 and 3.2.12) while r_s is the sampling radius (a constant). By allowing the angle to be any number in its domain, the sampled signal is found.

$$S_{sampled}(k) = S_{sampled}(k,t) \tag{3.2.19}$$

where, $t \in [0, 2\pi]$.

Finally, having both ideal and deformed signals of a particular pattern (with ID equal to k), by computing the similarity the effect of error in center of pattern can be seen. The error is a two dimensional vector therefore the self-similarity is a function of e_x and e_y:

$$ss(k,e_x,e_y) = \eta\left(S_{ideal}(k), S_{sampled}(k)\right) \tag{3.2.20}$$

Where η is defined in equation 3.2.7 and k is the ID of pattern.

Figures 3.9 and 3.10 depict the self-similarity criteria of some IDs. In these figures, $(e_x,e_y) \in [-5,5] \times [-5,5]$ is the domain (horizontal plane) and the value of

self-similarity function is shown on the third dimension. The range $[-5, 5]$ corresponds to $[-r_1, r_1]$ while in practice $e < r_1$, also in our case r_1 is equal to 5 pixels. In addition, as we expected before, for $(e_x, e_y) = (0, 0)$ the self-similarity is 1, because there is no error. However by introducing error, the similarity decreases with different rates based on the direction in which the center is shifted. This figure changes from ID to ID, and can be used as a criteria to find the sensitivity of that particular ID towards central shift error. For example, a sharper slope means more sensitivity to error.

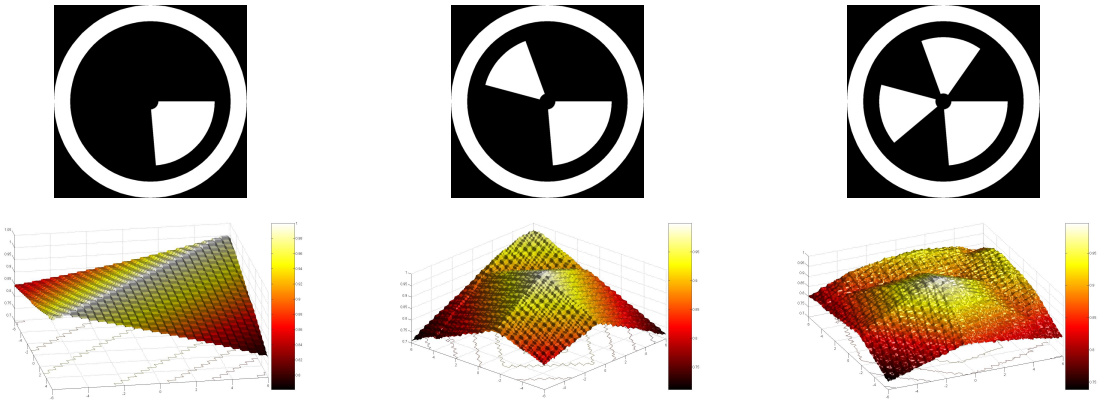

Figure 3.9: From left to right, a) ID equal to 1 b) ID equal to 3 c) ID equal to 6 The second row is self-similarity of first row (Z axis). X and Y axes are e_x and e_y.

Some conclusions can be achieved by looking at these 3D pictures; as Figure 3.9 shows for ID equal to 1, shifting of the center in a particular condition does not deform the signal, this can justify the robustness of this ID in practice and together with the distinctness of this ID (seen in Fig 3.6) candidates this ID as one marker. Although Figures 3.9 and 3.10 give some information about robustness of IDs, the robustness of heading is not conveyed. For example for the ID equal to 3 in (Fig. 3.9(b)) when the center is shifted towards the small white pie slice (β_4), self-similarity rises (to near

1) again, but in that case the heading is reversed. Furthermore, the self-similarities of IDs equal to 5 and 7 (not shown in this text), resemble those of IDs 1 and 4 respectively (only with some rotation).

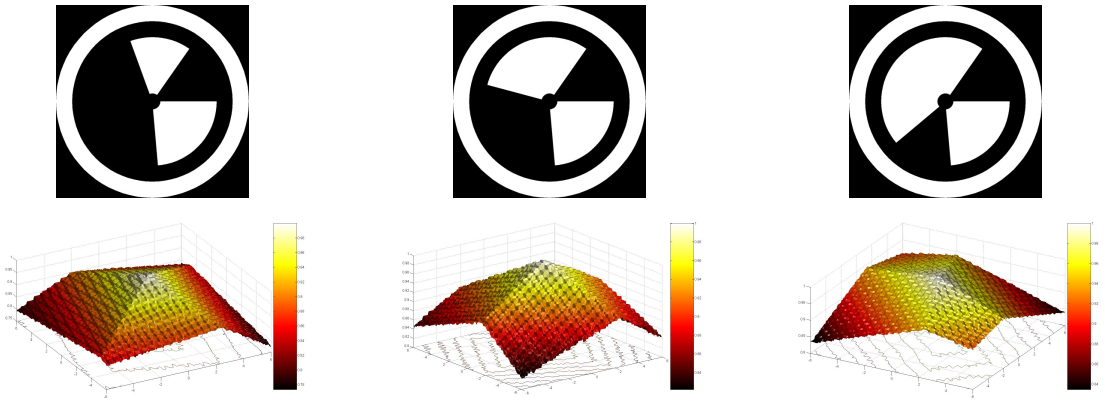

Figure 3.10: From left to right, a) ID equal to 2 b) ID equal to 4 c) ID equal to 8 The second row is self-similarity of first row (Z axis). X and Y axes are e_x and e_y.

3.3 Other patterns

We implemented the technique described for pattern identification in section 3.2, however the current system enables applying another methods as well. We used ImageJ as a standard package written for Java that has many features. We used this image processing package in *blob detection* (known as particle analyzer in ImageJ), enhancing the contrast and features such as zooming on the picture. Alternatively, we could for example reapply the blob detection algorithm within each found blob by applying *region of interest* (ROI) and find ID of the patterns by number or size of the inner blobs. The advantage of this approach would be more robustness towards deformation error, however finding headings will not be straight forward. By

looking at distinctness of IDs (Fig. 3.6) and also the self-similarity measurement (Fig. 3.9) some useful information could be achieved, for example, IDs 3 and 6 can be distinguished well but IDs 2 and 4 look similar. This leads to the assumption that because there is some overlap in the graphic of the latter group they might have a closer similarity. Based on this assumption another pattern that came into mind was 8 angle uniform circular pattern (8UCP). In the new design we place two always-black pie slice in between the angles x_1,x_2 and x_3 in the equation 3.2.3 . Note that the dimension 6 of vectors in the definition 3.2.1 is not empirical and we can expand it to define 8UCP.

Definition 3.3.1. An 8-angle-uniform-circular-pattern (8UCP) is defined by three vectors and one coding formula:

$$P_8 = \{\theta = [\theta_1, \cdots, \theta_8], \beta = [\beta_1, \cdots, \beta_8], \alpha = [\alpha_1, \cdots, \alpha_8], \phi, r_1, r_2\}$$

where,

θ is a vector of angles of 8 pie slices. such that:

$$\sum_{1}^{8} \theta_i = 360^\circ \tag{3.3.21}$$

$\beta_i \in \{0, 1\}$ for $i = 1, 2, 3, 4, 5, 6, 7, 8$. β is the color vector so that if β_i=1 the ith pie slice is white otherwise it is black.

α is the vector of position exponents(α_i are natural numbers or zero, for $i = 1, 2, 3, 4, 5, 6, 7, 8$)

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

$$\phi = \sum_{1}^{8} \beta_i \times 2^{\alpha_i} \tag{3.3.22}$$

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).

The improved instance of the equation 3.2.3 now is based on definition 3.3.1 as follows:

$$\begin{aligned} \theta &= [80, 55, 45, 5, 45, 5, 45, 65] \\ \beta &= [1, 0, x_1, 0, x_2, 0, x_3, 0] \\ \alpha &= [0, 0, 1, 0, 2, 0, 3, 0] \end{aligned} \tag{3.3.23}$$

This instance of pattern is basically expanding the equation 3.2.3 by adding two 5 degrees black positions between the informative part (which builds the code). In the next step since the definition of η would remain the same (Equ. 3.2.7) we can build a similar table of mutual similarities of the new IDs made by this pattern.

ID	1	2	3	4	5	6	7	8
1	1.0000	0.7354	0.7418	0.5707	0.7418	0.5707	0.5753	0.4448
2	0.7354	1.0000	0.4821	0.7761	0.4569	0.7761	0.3153	0.6048
3	0.7418	0.4821	1.0000	0.7694	0.4878	0.3276	0.7755	0.5996
4	0.5707	0.7761	0.7694	1.0000	0.4873	0.5074	0.5324	0.7793
5	0.7418	0.4569	0.4878	0.4873	1.0000	0.7694	0.7755	0.5996
6	0.5707	0.7761	0.3276	0.5074	0.7694	1.0000	0.4994	0.7793
7	0.5753	0.3153	0.7755	0.5324	0.7755	0.4994	1.0000	0.7731
8	0.4448	0.6048	0.5996	0.7793	0.5996	0.7793	0.7731	1.0000

Table 3.2: Mutual circular cross covariance of 8 angular patterns

Again the main diagonal of the table are all 1 because each ID has 100% self-similarity when there is no noise. The criteria in choosing distinguishable IDs is the same as before. The Fuzzy relation defined in 3.2.3 remains the same and figure 3.11(a) shows the new membership function based on 8UCP. Again darker cells in this figure are better options to choose because of more distinctness. By comparing figure 3.11(a) with 3.6, we can see more number of darker cells in the new pattern (8UCP). That suggests an improvement over the previous pattern, however the improvement is not significant. Especially the sensitivity to error in finding the center

remains the same(Fig 3.11(b)). In addition, using some techniques in image processing (like enhancing the contrast before blob detection and intelligent decision making) resulted in an acceptable and robust pattern recognition of 6UCP and eliminated the need to implement new patterns.

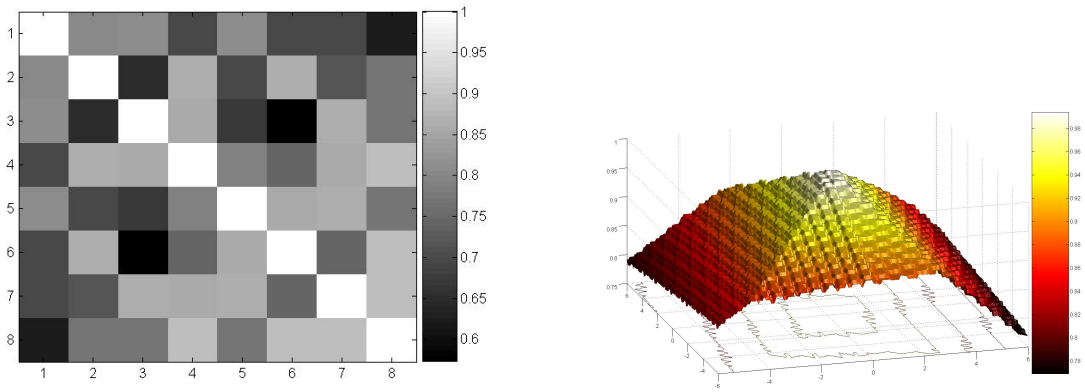

Figure 3.11: From left to right: a) Similarity membership function of for 8UCP b) self-similarity of ID=2 (Z axis). X and Y axes are e_x and e_y.

3.4 Image processing algorithm

Looking into the modeling aspect of the project, image processing is the vision of the robots, it should get information about the robots separately, and pass it to the real-time simulation being in progress. Image processing is implemented in a class called *Vision* which is the name of its functionality. Using the *Control Panel* it is possible to perform vision task only once or consecutively, the latter is done inside a thread. One important aspect in control is the sampling rate of the sensory input, when the input is in the form of captured images this is known as *frame per second* (FPS). Desired FPS is one of the settings associated with the interval of the vision thread,

and can be changed (Fig. 3.12). Note that because the vision task is time consuming, satisfying the desired FPS is based on the system resources such as speed and available cache. In addition, we perform vision in an efficient way, for example, the GUI (in which the on-line processed video stream is shown) is another thread which decouples the vision task from demonstrating the results. The result was about 10 FPS for a 1.8GHz Intel processor, for better results a good solution would be implementing image processing in a faster language (like C++) and connect it to Java using native codes. Furthermore, we assume parameters such as brightness, contrast, saturation and hue are fine tuned by the user in the GUI implemented for vision settings (Fig. 3.12). The vision algorithm consist of the following steps:

1. Build the image.
2. Convert the image to gray scale.
3. Enhance the contrast.
4. Find objects by Blob Detection.
5. Analyze object patterns.
6. Decision and mapping.

The first step of vision is responsible for preparing the image to be processed. First of all, the raw image data is in the form of a byte array representing the standard bitmap of the frame taken. This format is incompatible with standard Java image, therefore each element of the raw array is converted to a four byte integer (for Alpha Red Green Blue (ARGB) format). In addition, the sequence of the rows in the image is taken care of, if not the image will look upside down. Another additional conversion

is done to build the image compatible with ImageJ (known as ImagePlus) because the second, third and fourth steps of Vision are done using ImageJ. Apart from being available in Java, optimized, open-source and widely used, an advantage of ImageJ is the stand-alone software containing all of its features. It is easier to test applying different operations on still images (which can be saved as JPG and BMP format inside the control panel (*Save as BMP* and *save as JPG* buttons in Fig 2.4)), and after coming to a good method, write the necessary code to do the same procedures.

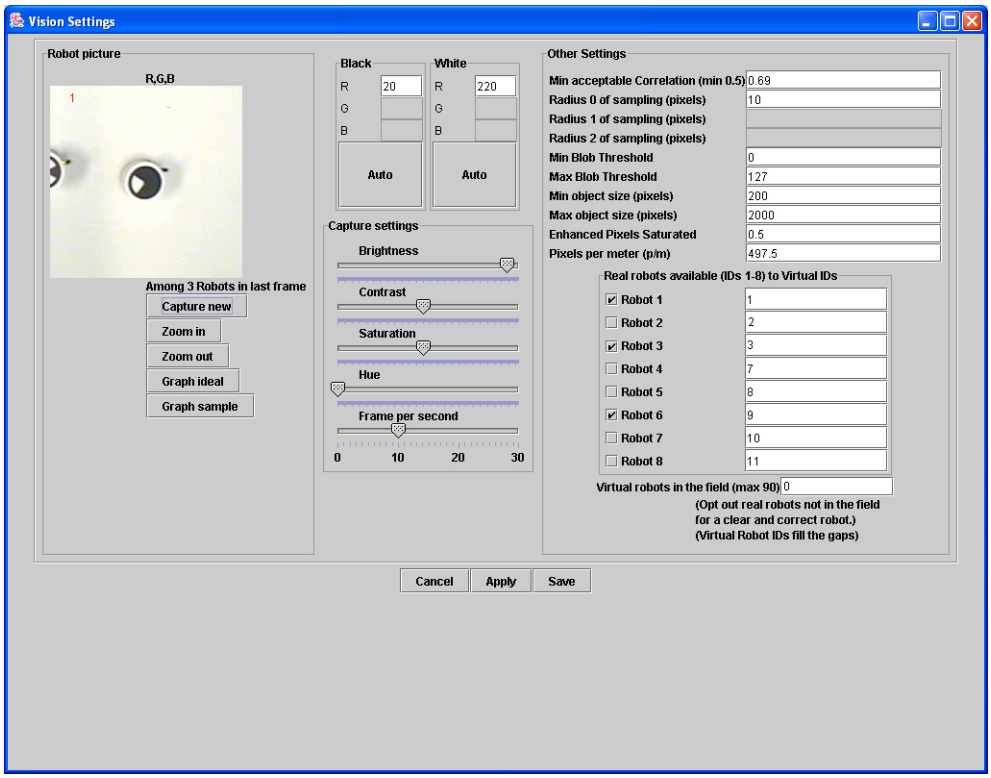

Figure 3.12: Vision settings GUI

In the second step of Vision, the RGB image is converted to gray-scale format (internally by averaging three bands and adding a constant offset of 0.5 to the result).

To have a sharper image in blob detection, the histogram of the image is stretched first. The saturation is controlled by *Enhanced Pixel Saturated* parameter (inside other-settings panel in Fig. 3.12) which is normally set to the half.

The fourth stage of Vision uses particle analyzer of ImageJ to find the blobs in the already prepared gray-scale image. The internal algorithm of particle analyzer works as follows. For each line of the image and each pixel, if the pixel value is inside the threshold range (given in Fig. 3.12 under *Min Blob Threshold* and *Max Blob Threshold*), 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. 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. Another filtering then is applied to opt out the objects which are not of preferred size (given in Fig. 3.12 under *Min Object Size* and *Max Object Size*). This filters some noise particles because we have some estimation about the expected objects' sizes in the field. We denote the total n objects with $\{o_1, o_2, \cdots, o_n\}$. Note that in general case n can be zero or more. Figure 3.13 shows totally five objects ($n = 5$) in the field, three of which are robots.

The result centroid measurement is then used as the center of possible 6UCP patterns in the place of each object (using the technique discussed in section 3.2). The result of pattern analysis is a table with each row representing an object mapping the similarity of that object to the markers (each with a specific ID) available in the field (*object to ID similarity mapping*). The similarity is measured by equation 3.2.7 for sampled signals and ideal signals.

As stated earlier, 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 $\{ID_1, \cdots, ID_m\}$. Figure 3.12 shows a panel named *"Real robots available"*, we call this panel, *mapping panel*. By selecting from the checkboxes in this panel user defines the available markers on the field, so that the *object to ID similarity mapping* would have fewer columns, this restriction proved to be very useful in the decision step (next step of algorithm). For example in Figure 3.12 only three markers are selected ($m = 3$) with IDs equal to 1,3 and 6 for three real robots in the field. These marker IDs are chosen based on criteria such as distinctness discussed in previous section (section 3.2). Table 3.3 shows the mapping of the five objects in figure 3.13 to three ($m = 3$) available marker IDs (named ID_1, ID_2 and ID_3 so that $ID_1 = 1$, $ID_2 = 3$ and $ID_3 = 6$) in one example frame of the video stream. Note that these values change based on many parameters such as error or ambient light over consequential frames.

	$ID_1 = 1$	$ID_2 = 3$	$ID_3 = 6$
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 3.3: Object to marker-ID similarity mapping in one frame data

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 object to ID assignment algorithm should find the

solution set Sol that:

$$Sol = \{(ID_i, o_j) | \, i \in \{1, \cdots, m\}\}$$
$$\text{if } (ID_k, o_l) \in Sol \text{ then } (\nexists j \neq k; (ID_j, o_l) \in Sol)$$

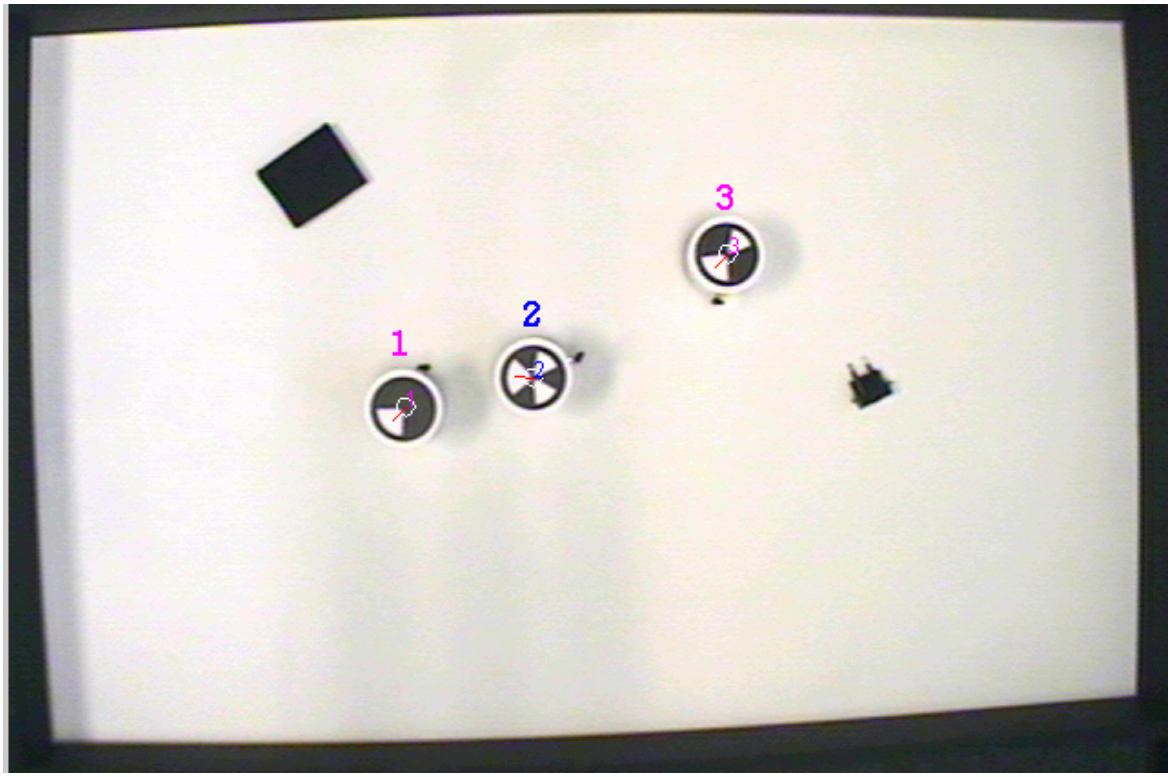

Figure 3.13: Platform and Mapping

There is a parameter known as *Min acceptable Correlation* also changeable in vision setting (Fig. 3.12), this parameter is the acceptable similarity threshold so that no object with similarity less than this threshold is assigned as valid robot in the algorithm that follows. Also note that for example table 3.3 a threshold of 69 percent is enough to eliminate two non-robot objects.

Since each object has a sampled signal, the similarity in the table is a result of maximizing cross covariance function, and $\eta(o_i, ID_j)$ is the similarity (η being defined

in equation 3.2.7). The search builds the solution set (Sol) iteratively by adding or removing some member elements. At first no object is related to any robots. Starting from $i = 1$, in each iteration for o_i, for all possible j the ID with maximum similarity is found (ID_g) if it is not related to previous objects, the algorithm assigns the ID ($=ID_g$) to the object (o_i) and goes to the next iteration by increasing i. If ID_g on the other hand is previously related to another object o_l ($l < i$) with more mutual similarity than new pair ($\eta(o_l, ID_g) \geq \eta(o_i, ID_g)$) this algorithm is repeated (without increasing i) with the next similar ID (searching while ignoring ID_g). The remaining case occurs when the newly found pair-similarity is more than among previously related pairs ($\eta(o_l, ID_g) < \eta(o_i, ID_g)$) in this case the algorithm removes (o_l, ID_g) from the solution and tries to relate the object o_l to possibly another ID. The steps of the *decision algorithm* which is done for each captured frame follows:

1. start by $i = 1$

2. $(o_i, ID_g) = argmax_j \left(\eta \left(o_i, ID_j\right)\right)$

3. if $\nexists j \in \{1, \cdots, i-1\}$ that $(o_j, ID_g) \in Sol$

   then add (o_i,ID_g) to Sol,

   if $i < n$ then terminate the algorithm else increase i and goto step 2 (n is the number objects).

4. if $\exists l \in \{1, \cdots, i-1\}$ that $(o_l, ID_g) \in Sol$ and $\eta(o_l, ID_g) \geq \eta(o_i, ID_g)$

   then set $\eta(o_i, ID_g) = 0$, and goto step 2

5. if $\exists l \in \{1, \cdots, i-1\}$ that $(o_l, ID_g) \in Sol$ and $\eta(o_l, ID_g) < \eta(o_i, ID_g)$

   then remove (o_l, ID_g) from Sol,

set $\eta(o_l, ID_g) = 0$, set $i = l$ and goto step 2

The solution set for the example table given in Figure 3.3 is $\{(o_2, 3), (o_3, 6), (o_5, 1)\}$. Furthermore, we should mention that, all of the vision steps except the two last steps are implemented as separate functions. So far we have not used other off-line data such as the similarity between IDs themselves (Table 3.1), a good future research problem would be to consider this data as follows. Based on the definition 3.2.3 we can use only a subset of F (definition. 3.2.3) but only for the available IDs.

Definition 3.4.1. The Fuzzy subset for m specific IDs is defined as:

$$F_a = \{((i,j), \mu(i,j)) \in F \,|(i,j) \in U_a\}$$

where,
U_a=$\{ID_1, \cdots, ID_m\} \times \{ID_1, \cdots, ID_m\}$ is the universe ($U_a \subseteq U$)
$\mu(i,j)$=$\mu_F(i,j)$ is the membership function.
F, μ_F and U are defined in definition 3.2.3

For example the Fuzzy subset of IDs in $\{1, 3, 6, 7\}$ is shown in the table 3.4 (which can also represent a matrix).

ID	1	3	6	7
1	1.0000	0.7418	0.5707	0.5753
3	0.7418	1.0000	0.3276	0.7755
6	0.5707	0.3276	1.0000	0.4994
7	0.5753	0.7755	0.4994	1.0000

Table 3.4: A Fuzzy subset of similarities

By observing the tables 3.1 and 3.4, and also Figure 3.6 and also recalling the pattern analysis, we can see that for example the ID pairs (3,7) are much more similar than any pair of (1,7) and (6,7). This information can be used as another distinctive

feature for ID equal to 3. Here ID equal to 7 is not an available ID, however we can assume it is available and build a table similar to Table 3.3. The result extended table is like Table 3.3 with an additional column (the left matrix in Equ. 3.4.24). (Here in this example, we call 7 a support ID for 3 while 3 is the owner of 7). This extended matrix is a Fuzzy relation between objects and available IDs (union with the support IDs), where the membership function gets its values from the object-ID mapping table cells, and the universe is the possible combination of objects and available IDs union support IDs. If we call this extended Fuzzy set F_o, after replacing the support IDs with their owners, the solution set is $Sol \subseteq U_{F_o}$ while F_o (with universe U_{F_o}) is the search space of the mapping decision algorithm in image processing. To further utilize these mutual similarities we can combine the relations F_a with F_o as a filter.

$$F_o \leftarrow F_o \circ F_a$$

$$F_o \leftarrow \begin{bmatrix} 0.2743 & 0.3336 & 0.4039 & 0.3703 \\ 0.6221 & 0.9315 & 0.1633 & 0.6984 \\ 0.5072 & 0.1972 & 0.9139 & 0.4102 \\ 0.3381 & 0.4781 & 0.3691 & 0.5073 \\ 0.9299 & 0.6591 & 0.5136 & 0.5149 \end{bmatrix} \begin{bmatrix} 1.0000 & 0.7418 & 0.5707 & 0.5753 \\ 0.7418 & 1.0000 & 0.3276 & 0.7755 \\ 0.5707 & 0.3276 & 1.0000 & 0.4994 \\ 0.5753 & 0.7755 & 0.4994 & 1.0000 \end{bmatrix} \tag{3.4.24}$$

The composition can be done using the S-Norm ($\perp_{max}$) and T-Norm (T_{prod}) for Equation 3.4.24. Now the same 5 step decision algorithm can be applied on the new F_o (Equ. 3.4.25) to find the solution set and then finally substituting the support IDs (here only 7) with their owners (here just 3), if they appear in the solution set.

$$F_o \leftarrow \begin{bmatrix} 0.2743 & 0.3336 & 0.4039 & 0.3703 \\ 0.6910 & 0.9315 & 0.3550 & 0.7224 \\ 0.5216 & 0.3762 & 0.9139 & 0.4564 \\ 0.3547 & 0.4781 & 0.3691 & 0.5073 \\ 0.9299 & 0.6898 & 0.5307 & 0.5350 \end{bmatrix} \tag{3.4.25}$$

The result of image processing is a set of robot objects with tags such as position, IDs, heading and similarity to ideal robot. These IDs are numbers associated with real robots based on the markers put on top of them. These real IDs therefore can be any number between 1 and 8 ($ID_r \in \{1, 2, \cdots, 8\}$) , on the other hand in the robot convoy problem when real robots and virtual robots work together we need a set of increasing numbers given to a mixture of real and virtual robots (which are called agents together). Agents' IDs can be a set of natural numbers, of course having more elements than number of real robots in the field, to cover real and virtual robots ($ID_v \in \{1, 2, \cdots, T\}$ T being the number of agents). The mapping of real IDs (ID_r) to their virtual counterparts (ID_v) can be done in vision setting GUI (Fig. 3.12) in *mapping panel* because this mapping is only valid for available IDs which is defined in the same place using checkboxes.

In this mapping, the desired virtual IDs for given available real IDs are given (in the GUI), the mapping function tries to associate the virtual robots to the other virtual IDs available to satisfy the desired virtual IDs of real robots, for those IDs for which desired mapping is not possible an automatic mapping is performed. The numbers shown in figure 3.13 are the agents' IDs, agents with the same virtual and real ID (3 and 1 in this figure) are shown in the same colors and different from other agents. We can say that the IDs 1 and 3 are mapped to themselves, but 6 is mapped

to number 2. We have the condition $1 < 2 < 3$ satisfied among the result of this mapping, therefore the order in which the agents follow each other is specified.

Finally, based on the modularity, ID_r is used for vision related side of the application, while the real-time simulation works with ID_v. Finally, the decision making part is an open research problem, where many techniques such neural networks, Kalman filtering and Bayesian networks can be used. Especially since we did not use the localization results of previous frames when localizing robots in current frame.

Chapter 4

Robot-in-the-Loop

The software development methodology if based on the simulation, can solve the *incoherence problem* between different stages of the process by applying the *model continuity*. This technique makes use of simulation to test the real-time embedded software effectively, where virtual models replace some physical objects and sensors during the test phase. The step-wise simulation based software testing and development, enhances the technique to examine the logic and temporal characteristics of the real-time system [13]. The *dynamic team formation* is a test-bed to further investigate the step-wise refinement of the software by simulation (*Robot-in-the-loop*). Whatever architecture chosen for problem solving of the distributed software, this methodology can help greatly in fine turning of smallest units and building blocks. Choosing behaviors as the building blocks is a common technique yielding robustness in the dynamic environment of multi-agent team formation [25, 36].

Considering the robot swarm problem [24, 37, 25] where many micro-robots (like ants) are deployed for a mission, to decompose the mission goal into simple behavior tasks is often challenging [24]. On the other hand, behaviors provide a bottom-up solution with an easy way to think of the problem. Unlike the common subsumption

behavior structure (with hierarchical behaviors) which is often used in multi-robotics, we base our architecture on mutual inhibition and DEVS (introduced in introduction section). While the *context layer* gives the desired effect of multi-layer hierarchy, putting the behaviors in the same level, by applying the *Robot-in-the-loop* software design, it is possible to systematically revise the behavior tasks. In addition, unlike most of the previous works involving simulation and multi-robot [25], the computer code used to drive the simulation is the same as one used for real robots. Like *Human-in-the-loop* interactive simulation [38, 39], the idea of *Robot-in-the-loop* as a paradigm of *model continuity* is to close the gap between reality and simulation by solving the *incoherence problem* between several stages of development [13, 40, 41].

4.1 Dynamic team formation

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 $k \in \{2, 3, \cdots, N\}$. The final team would be $\{R_1, R_2, \cdots, R_N\}$. 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 ($\{R_l, R_{l+1}, \cdots, R_m\}$ 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 ($l \leq o < p \leq m$), 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. Furthermore, the first agent R_1 is unique because it does not have to search for any other agent, and leads the entire group. In addition, R_1 does not follow any specific pre-defined path which gives it more free movement. To model the agents, we assume the agents are homogenous (which is true for virtual robots but not completely true for real robots even of the same brand).

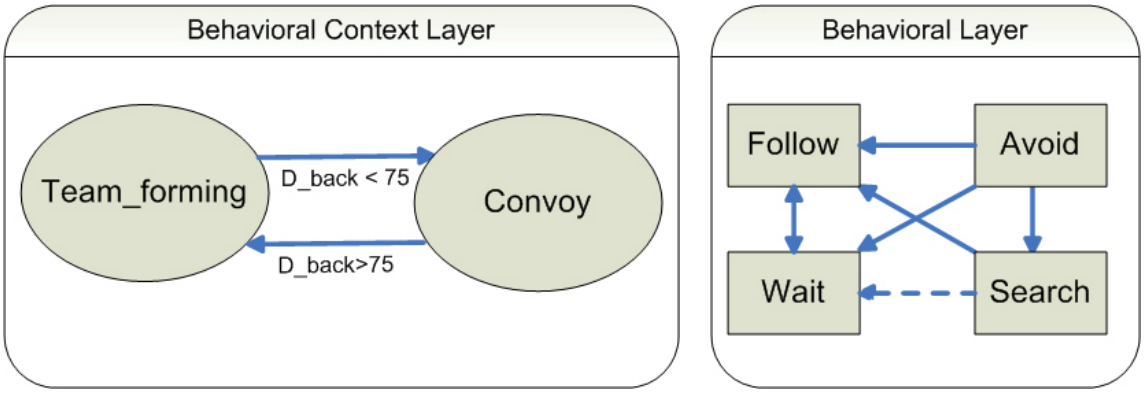

Figure 4.1: From left to right, a) The behavioral context transitions b) Behaviors and their inhibitory connectors

The behavioral context (Fig. 4.1(a)) of the robot-convoy problem has two states for each agent; *Team_forming* (which is during the team formation) and *Convoy* (after sub team formed) these states transit to each other according to *D_back* ($D\_back = \min(d_5, d_6)$ where d_5 and d_6 in Fig. 4.2 are defined later). By considering the possible behavior functionalities such as motion, orientation, navigation, clustering and dispersion [24] we chose four behaviors; *Follow*, *Avoid*, *Wait* and *Search*.

Figure 4.1 shows the behavioral and context layers of the team formation problem. Having these behaviors we only need to define the excitation of each behavior (Equ. 1.2.2) and the inhibitory coefficient matrices corresponding to each context to

find the inhibition applied (Equ. 1.2.1). Afterwards, the task of each behavior would be defined and decision making algorithm is complete. The constant thresholds for firing actions are $T_{inhibit} = 0.1$ and $T_{exe} = 0.2$. The excitation and actions are based on the sensory input from virtual sensors.

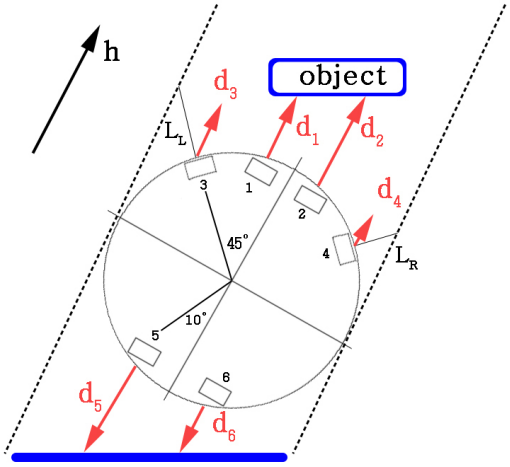

Figure 4.2: Proximity sensors model of the agent

Figure 4.2 shows the typical plan for 6 out of 8 Khepra® infra red proximity sensors [32], their relative positions and functionality in sensing front and back side of the robot. These sensors give a straight distance of the objects found in front of them; which for the sensors 1, 2, 5 and 6 it would be $\{d_1, d_2, d_5, d_6\}$. As shown in this figure, the measurement is done parallel to agent's heading (h) to find the objects in the course of the agent (as obstacles). Therefore the measurements of the sensors 3 ($= D_L$) and 4 ($= D_R$) are projected to h vector to find d_3 and d_4 only if there is any object between the dotted lines. In another words, if the object is placed further than L_R it is not detected by sensor 4 and as a result d_4 would be infinity, measuring d_3 is done similarly.

$$d_3 = D_L \times \cos(45) \text{ if } D_L \leq L_L \text{ otherwise } \infty$$
$$d_4 = D_R \times \cos(45) \text{ if } D_R \leq L_R \text{ otherwise } \infty$$

Table 4.1 shows the excitation formula for each behavior of the the agent R_k $(1 \leq k \leq T)$, the actions of behaviors come in the following sections. The explanation of the *Avoid* behavior is based on figure 4.2 which is the extended model of the agent sensors, required in section 4.4, before that stage only two sensors were considered in modeling, one in front of the agent and one at the back, each calculating d as the distance to the obstacle.

	Excitation	Explanation
Avoid	$E_{avoid} = \min\left(e^{-\frac{d-20}{10}}, 1\right)$	$d = \min(d_1, d_2, d_3, d_4)$
Follow	$E_{follow} = \begin{cases} 0.9 & \text{if } k = 1 \\ \min\left(0.4 + 0.6 \times e^{\frac{d-100}{10}}, 1\right) & \text{o.w.} \end{cases}$	d=Distane to R_{k-1}
Search	$E_{search} = \begin{cases} 0.8 & \text{if } d > 150 \text{ and } k \neq 1 \\ 0 & \text{o.w.} \end{cases}$	d=Distane to R_{k-1}
Wait	$E_{wait} = \begin{cases} 0 & \text{if } k = T \\ \min\left(e^{\frac{d-120}{10}}, 1\right) & \text{o.w.} \end{cases}$	d=Distane to R_{k+1}

Table 4.1: Excitation of the behaviors of the agent R_k $(1 \leq k \leq T)$

Figure 4.3 shows the effective range of different behaviors in general, where E_A,E_W,E_S and E_F represent the excitation of the behaviors *Avoid*, *Wait*, *Search* and *Follow* respectively (in table 4.1). According to the original design [12], each agent has two stages; at the beginning of the team formation it should *search* for the front agent more aggressively then *follow* it without waiting much for the back robot. However, after the back robot comes to a certain distance, the agent should favor *waiting* over *searching* and *following*. The former phase is the *Team_forming* context and the latter

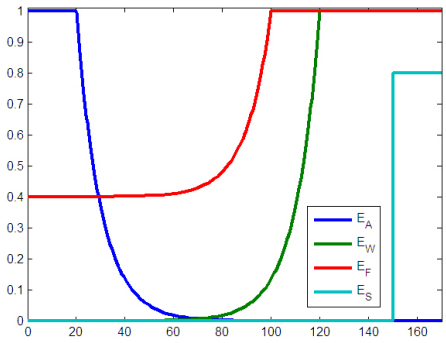

Figure 4.3: Excitations effective range

is the *Convoy* context.

Tables 4.3 and 4.2 show the inhibitory coefficients of the behavioral network in two different contexts [12]. In these tables, the behavior in the upper row is inhibiting the behavior in the left column. Furthermore as can be seen in among the inhibitions, the priority of *search* and *wait* is reversed during the context change; in the *convoy* context *wait* behavior inhibits *search* (with coefficient of 0.6) while in the *team forming* context *search* behavior inhibits *wait* with the same power. The relative priorities of the behaviors *follow* and *wait* also change to some extent, but the behavior *avoid* still inhibits all other behaviors with the highest degree (1.0) because it should always avoid the collisions.

Finally, because of the robustness of the algorithm it is observed that most of the constants used for inhibitory coefficients (Tables 4.3 and 4.2) and also excitation formulas (Table 4.1), are not empirical values and can be changed within a large range as long as they conserve the concepts such as relative priorities of the behaviors and the time they should be fired.

		Inhibiting			
		Follow	Wait	Search	Avoid
Inhibited	Follow	x	0.5	1.0	1.0
	Wait	0.6	x	0.6	1.0
	Search	0.0	0.0	x	1.0
	Avoid	0.0	0.0	0.0	x

Table 4.2: Team forming context, mutual inhibitory coefficients

		Inhibiting			
		Follow	Wait	Search	Avoid
Inhibited	Follow	x	0.7	1.0	1.0
	Wait	0.4	x	0.0	1.0
	Search	0.0	0.6	x	1.0
	Avoid	0.0	0.0	0.0	x

Table 4.3: Convoy context, mutual inhibitory coefficients

4.2 Test the design as fast as it can

The architecture of the system is described in previous sections. The iterative testing of the design is done using simulation. As stated earlier the real and virtual robots are modeled to work together in simulation framework. However, before bringing the real objects in the simulation there is no need for real-time simulation because the convoy can consist of only virtual robots which try to form a team based on the behavior based modeling architecture.

Since this phase of the robot-in-the-loop, involves non real-time simulation, the simulation is done as fast as the computer speed allows. In another words, because the virtual time does not need to be synchronized with world clock, the sequence of events occur sooner than expected time. Another result is that the virtual time

interval between two consequential events does not need to be consistent during the simulation. The latter result often will not have a tangible effect if the amount of calculations between two consecutive events is not too much, which is often the case.

	Task
Avoid	move backward for $[10, 20] \rightarrow$ rotate clock wise for $[60, 80]$ degrees. Using non resumable task queue.
Follow	turn to agent $R_{k-1} \rightarrow$ move forward or backward to keep desired distance ($d_{desired}$) of 20. Using non resumable task queue.
Search	move forward for $[30, 50] \rightarrow$ rotate counter clockwise for $[70, 80] \rightarrow$ move forward for $[30, 50] \rightarrow$ rotate clockwise $[70, 80]$. Using resumable task queue.
Wait	Null

Table 4.4: Initial task definition of the behaviors of R_k

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 old 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 under different initial conditions such as different size, number, position and heading for the agents.

For the current problem the robot-in-the-loop is required to help us design the tasks of the behaviors. Table 4.4 shows the initial design for the tasks of each behavior [12]. Most of the constant numbers are not empirical and can vary as long

as they preserve the meaning of the behaviors, also the angles are in degrees but the distances do not have any unit now (later in the real-time simulation the distances are considered as the number of pixels and converted to the actual distances in meter). In addition, navigational commands such as *turn to*, *rotate* and *move* are considered to be available as well as the distance sensor data. In this notation the task is defined for the k^{th} agent R_k. The notation $[n_1, n_2]$ means a random number between n_1 and n_2 and arrows show the sequence. The tasks in this design are considered to be run in task queues, resumable or non-resumable. Furthermore, we have $d_{desired} = \min(d_1, d_2, d_3, d_4)$, but the individual distances are not exploited yet. Now the simulation on all virtual agents (in Fig 2.5) is used to test this behavior task design, for validity in the long run.

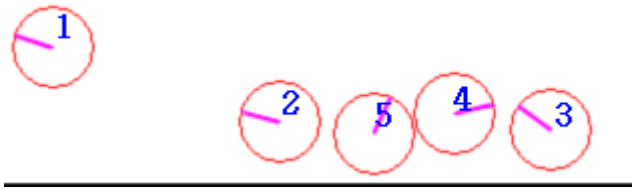

Figure 4.4: A simple deadlock caused by limited space and lack of movement

As discussed earlier the behaviors individually act like finite state machines, thus certain configurations may lead to deadlocks without a mechanism to bring the system out of the deadlock. In earlier designs of the sensor model of the agents, only two sensors were considered; one in front and one at the back side of the robot. This leads to over abstraction by ignoring the space one robot takes and as a result agents may collide and even pass over each other while in reality this is not possible. On the other hand the more realistic sensor model (Fig. 4.2) restricts the free movement of

agents as is common in reality. This restriction together with the FSM-like behavior-tasks cause the simulation to freeze in a certain cases after running the simulation for certain period of time and certain number of agents. Figure 4.4 shows one of the simplest forms of this area-bounded deadlocks.

The deadlock in figure 4.4 is caused by arrangement of the agents so that the agent R_1 is waiting for R_2 and R_2 is waiting for R_3, but R_3 can not move forward, since R_4 is placed in its desired distance with R_3 and does not like to move. Also R_3 is constrained by a wall in the bottom side. This deadlock is not stable and can be solved easily by adding more activity (or temperature) to the tasks, the required energy can be added by applying more randomness and freedom of degree to the behaviors. It acts like heating the molecules of gas so they tend to widen in space. To achieve this, wherever there was an explicit rotation (clockwise or counter clockwise) we randomly change the direction therefore the commands *rotate clockwise* and *rotate counter clockwise* are replaced with the command *rotate* which has no explicit direction. Note that adding too much temperature (for example by increasing the span of the random rotation from $[70, 80]$ to $[0, 360]$) can result in erratic movements.

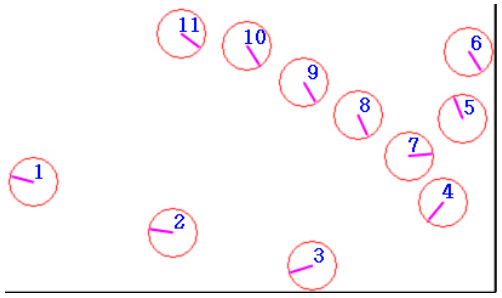

Figure 4.5: A more complex deadlock

The new design could solve much of the deadlocks, for instance R_2,R_3 and R_4 (in

Fig 4.4) now move more aggressively and finally find a way to continue. However, Because of the realistic design of the sensors, it is more likely to face deadlocks when the number of agents is increased, because the free movement is hindered by lack of space for maneuver. Having run the simulation for different configurations another non-resolvable deadlock was discovered (Fig. 4.5).

	Task
Avoid	move backward for $[10, 20] \rightarrow$ rotate for $[60, 80]$ degrees. Using non resumable task queue.
Follow	turn to agent $R_{k-1} \rightarrow$ move forward or backward to keep desired distance ($d_{desired}$) of 20. Using non resumable task queue.
Search	move forward for $[30, 50] \rightarrow$ rotate for $[70, 80]$ degrees $\rightarrow$ move forward for $[30, 50] \rightarrow$ rotate $[70, 80]$ degrees. Using resumable task queue.
Wait	increase suspended time (t_{sus}) if behavior has not changed

Table 4.5: Refined behavior-tasks of agent R_k

The added free maneuvering could not solve the deadlock in Figure 4.5. This deadlock is between the agents in the set $\{R_1, R_2, R_3, R_4, R_5, R_6\}$ while other agents such as R_7 act as barriers. By analysis of this deadlock again we see the combination of *avoid* and *wait* behaviors to be involved. R_5 is in its desired distance with R_4, thus not moving much, unless a little movement when R_6 approaches to pass it. The black lines are the environment boundary that agents can not pass. Since the deadlock happened as a consequence of unlimited *waiting* for previous agents, we added another parameter to the waiting behavior so that each agent has a limited *patience* while waiting for the follower agent.

$$E_{wait} = \begin{cases} 0 & \text{if } k = T \\ P \times \min\left(e^{\frac{d-120}{10}}, 1\right) & \text{otherwise} \end{cases} \quad (4.2.1)$$

$P = e^{-\frac{t_{sus}-15}{5}}$ where e is the the base of the natural logarithm

By decreasing the patience (P) over the time the agent which has waited more (in deadlock) tends to break the chain. For example R_3 in the latter deadlock starts moving forward running out of the patience. The excitation of the *wait* behavior thus changes (from what was defined in table 4.1) to what is defined by equation 4.2.1 for agent R_k. The t_{sus} used in calculation of patience (P) is the time agent (R_k) has been in waiting behavior (suspended time). When suspended time reaches about 15 seconds, the patience is decreased. If no other agent intrudes the waiting agents, the chain is more likely to break in the point of the deadlock thus still keeping some agents in the resulting sub teams (like $\{R_1, R_2, R_3\}$). The sub-teams continue their convoy, while agents try to form the complete team again. During the simulation agents may form a team then it is broken and formed again in a dynamic way, but simulation helped solving the deadlock problem.

4.3 Bring one real robot, synchronize the pace

In the next step in *robot-in-the-loop*, one real robot is added to the all-virtual convoy, this is the case when only one agent in AgentSys (Fig. 2.5) is a real robot and the rest are virtual. This step is a big change as compared to previous phase because we are adding to the dimensionality of the problem. One instant result is that because of the real object we need to switch to the real-time simulation, make changes to the design then go back to the previous phase test the overall logic and iterate this until a desired outcome is achieved.

In the case of robot-in-the-loop, the very first issues to solve was the virtuality of the units and scales in the DEVS simulation design. While the virtual robots can have any size, moving speed, and angular speed (as long as these parameters are relatively meaningful among the virtual robots), the equivalent scales must have some physical meaning for real robots. For example the speed should be in terms of meter per second so that real objects behave correctly. So far, it was possible to scale the environment and robot size for instance, and the simulation result was the same because the relative scales were held. To make the virtual and real objects work harmonically, we decided to apply the physical units like *meter*,*meter per second*, and *radian per second* for measurements and navigational commands (sice real robots can understand them). Having the actual size of the real robots and the platform, by considering a constant called *Pixels per meter* (which can be changed in vision setting (Fig. 3.12)), the scales and units were converted inside the interface (Fig. 2.5) for real robots.

After the units have meaning in real world, by choosing an appropriate time step for DEVS realtime simulation (so that events can be handled within each time frame), the DEVS real-time simulation can be started. The simulation time step can be changed in the main GUI (Fig. 2.4). After this correction, all agents should move with the same speed if they receive the same navigational command in the similar context.

While the *as-fast-as-it-can* simulation in DEVS does not require synchronization with real world clock, the real objects need this clock to function correctly, when interacting with other models. From the control aspect of the problem, the finer the

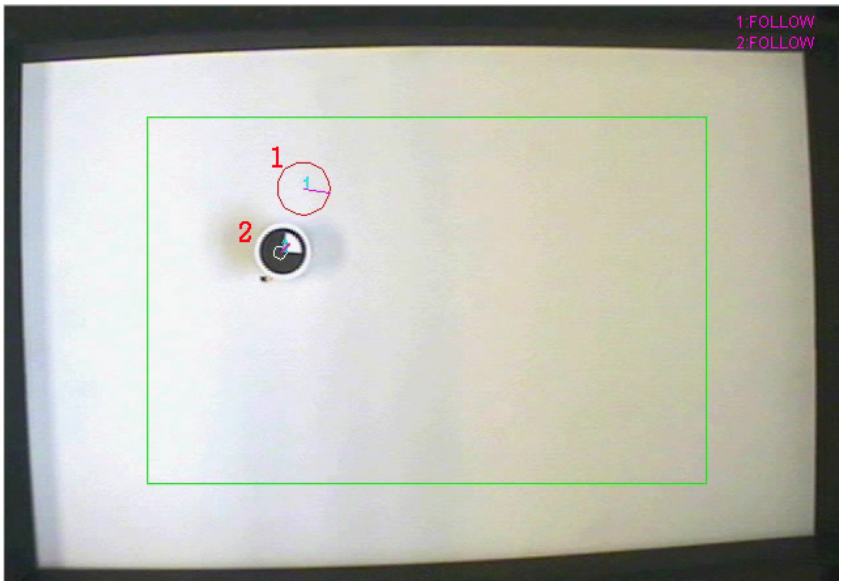

Figure 4.6: One real robot in the loop

time interval, the more controllability of the system is expected. However the computational power of the machine limits the time resolution selection, especially when heavy calculations such as real-time video processing of the input data is required.

When the first real robot comes to the simulation loop (*robot-in-the-loop*), the reality changes some design concepts that were taken for granted before, for examples the task of the behavior *wait* did not have any navigational commands (Tables 4.4 and 4.5) because doing nothing meant to stop and keep the previous place. For real robots on the other hand, if the navigational command does not change, the robot keeps running the previous command. To actually stop the robot another navigational command (*stop*) was added to the tasks of waiting. Using equation 2.5.2 to stop the robot we need to adjust the speed of both wheels to zero ($V_L = V_R = 0$), this is also done in the interface of agent with the environment (Fig. 2.5).

Other behavior tasks also were redefined to get more efficiency. The all-virtual case of robotic convoy simulation takes the DEVS inherent advantage (of being event based) to remove the actual time between events for a faster result. This is useful to observe the system's behavior in a long time and by ignoring the details, however it cannot help much in designing an efficient system because of virtually eliminating the effect of the speed of convergence. What may take hours in reality may pass in minutes during the virtual-time simulation because only the sequence and precedence of the events are taken into account. On the other hand, a faster team formation is required, because not only it is more desirable (in terms of efficiency) but also the real robots have limited battery charge and limited speed. Table 4.6 shows the new design for behavioral tasks fitting the case with one real and many virtual robots. The distances d_1, d_2, d_3, d_4, d_5 and d_6 are the sensory inputs of the extended sensor model of R_k having the heading of h (Fig. 4.2), In addition $d_{min} \in \{d_1, d_2, d_3, d_4, d_5, d_6\}$ is the distance to the nearest object.

The new design is tested in the DEVS realtime simulation (Fig. 4.6). As can be seen in this picture, one real and one virtual robot formed a convoy and both are in behavior *follow*, where the real robot (R_2) is following R_1. By changing the mapping of real to virtual IDs (as discussed in vision), the inverse case is also simulated where the virtual robot follows the real robot. In addition, the case with one real robot and many virtual one is simulated, too. During these experiments, new behavior tasks for *avoid* and *follow* were suggested (Table 4.6).

In previous best design (Table 4.5) every task was enqueued in a task queue, during the time a behavior was active this queue was used to get the next navigational command. The effect of these queues, regardless of being resumable or not, was a

	Task
Avoid	if $d_{min} \in \{d_1, d_2\}$ { rotate clockwise if $d_1 < d_2$; rotate counter clockwise otherwise if $d_{min} \in \{d_3, d_4\}$ { rotate clockwise if $d_3 < d_4$; rotate counter clockwise otherwise else move forward
Follow	turn to agent R_{k-1} if the angle between h and position of R_{k-1} is more than $\frac{\pi}{6}$ otherwise move forward or backward to keep desired distance ($d_{desired}$) of 20.
Search	move forward for $[30, 50] \rightarrow$ rotate for $[70, 80]$ degrees $\rightarrow$ move forward for $[30, 50] \rightarrow$ rotate $[70, 80]$ degrees. Using resumable task queue.
Wait	Stop, increase suspended time (t_{sus}) if behavior has not changed

Table 4.6: Refined behavior-tasks of agent R_k after bringing one real robot

series of navigational commands each having a specific duration. This design, was good as long as the simulation was fast and with virtual robots. During the real-time simulation on the other hand it was not efficient. With queuing, for example if R_1 was in P_1 and R_2 was in *follow* behavior, R_2 might have received a mission to go near P_1 while R_1 had changed the position to P_2 still in a range far enough to invoke and keep R_2 in behavior *follow*.

The problem which is discussed in the previous paragraph is the result of out-dated knowledge about the environment. Although this semi off-line decision making did not show up in the *as-fast-as-it-can* simulation, with slow real robots and real-time simulation this inefficient decision-making was revealed. The problem was more evident with the *avoid* behavior where immediate decisions were needed to avoid obstacles and virtual walls (the green border in the simulated area (Fig. 4.6)). Therefore the behaviors *avoid* and *follow* changed to the on-line version, while the behavior

search remains the same (Table 4.6). Using the individual sensor inputs, helps making a more accurate decision, while executing few navigational commands each time (instead of queuing them) gives the *follow* and *avoid* behaviors faster response to the dynamic changes, thus better performance and faster team formation. New behavior *avoid* for example is an online combination of turning and moving forward which is completely different from previous design.

4.4 More real robots, face the reality

The virtual robots do not crash if they collide with each other, the same is true with a real and a virtual robot collision. This is an obvious fact that makes the experiment with more real robots a need to test the system before accepting the model. With only two proximity sensors at the front and back side of the robots it is hard to avoid all collisions, therefore a new sensor model was taken into account to avoid agents collision (Fig. 4.2). Because of this drastic change to the control logic, based on the incremental design methodology, the simulation-based design should start from the all-virtual simulation again, leading to new results, that is why the previous behavior tasks are expressed in terms of the extended sensor model which is actually designed in this phase. Figure 4.7 shows two real and some virtual robots, in a following convoy (all in behavior *follow*).

In addition, the virtual robots are well behaved and identical (homogenous) but real robots are intrinsically heterogenous. While we can expect the same behavior from two virtual robots (when receiving the same command in the same context), real robots may behave differently because they may have different mechanical characteristics or be located in different positions of real environment. The real environment

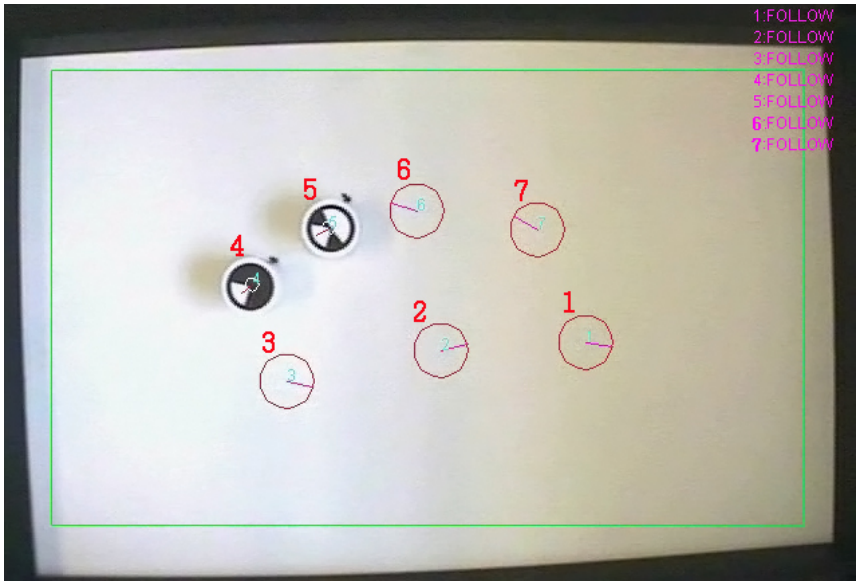

Figure 4.7: Two real robots in the loop

in our case is just a simple platform which can be assumed to be uniform in different locations (Fig. 2.2). For example by receiving the navigational command of rotation (which should ideally rotate the robot around its center based on equation 2.5.2), a real robot may have some displacement which is even different from another real robot. It should be mentioned that the real robots are homogenous in shape however (all rounded with the same radius), which justifies the equivalent modeling and simulation.

To reduce the effect of unexpected behaviors (thus having more robust models) of real robots. One way to handle this is to send compensatory commands in a closed loop manner, for example after sending a rotational commands if the robot has an unexpected displacement and the displacement is not in line with the course of robot another movement may correct it. If a robot, in behavior *follow*, is moving

away from the leader robot, then a displacement command (like moving forward or backward) can correct the rotational command's unwanted effect. On the other hand, sending corrective commands over the link increases the complexity of design (and also consumes more bandwidth as will be discussed in the next section). In the distributed team formation in which a robot itself is running an embedded application, this problem can be solved by a separate task of alignment while Khepra® can be programmed for several concurrent tasks [31]. Implanting the alignment task (to solve the unwanted displacement by rotation command), inside the on board program reduces the complexity of the modeling caused by heterogeneity.

4.5 More on complexity

The final system is a team formation with all real robots without being controlled by central DEVS simulation on a PC. It is the distributed team formation which is the next step of the project and is still under development in the time this text is being written. To move to that stage however, we needed to further bring real robots and simulate them to see if new concepts lead to new architecture (behavior tasks in this problem). Figure 2.4 shows the DEVS real-time simulation with three real robots after passing previous design phases.

Another issue, which becomes more obvious as more and more real objects are introduced to the real-time simulation, is the mutual effect of these objects on each other, the hidden effects and what was not predicted beforehand. The communication of DEVS models is through the ports of coupled models (Definition 1.1.2), as a result the virtual robots receive all of the commands on time independently. On the other hand to send the messages to the real robots we use a communication link

(Fig. 2.7), this imposes some restriction on the bandwidth of communication which is high enough to ignore its presence when sending command to few robots. Taking into account the inefficient wireless communication of the standard Khepra®, the interference of communication links of real robots and the packet loss, the more we bring real robots the more this problem shows up. This phenomena among the real objects illustrate the dependency of new steps in the design.

One way to tackle the bandwidth problem is to use a better wireless communication link or to avoid sending unnecessary information. To achieve the latter, the communication unit was changed to send a navigational command to a robot only if it has changed (although removing the redundant packets may ignore some commands in a noisy environment). In addition, the distributed software (which is supposed to work in distributed team formation), should also support the standard communication of Khepra® (SerComm) so that it can be used in central team formation too, while it will not send the echo of the commands (unlike the available software). By avoiding to send the echoes of the commands, the effective bandwidth will be doubled and also less interference will occur.

4.6 Incremental simulation-based design process

The incremental system development method discussed here has some partial outcomes in each phase, Table 4.7 shows the product of each step and what is expected to be achieved in that step. The design is incremental starting from the basic all-virtual simulation and incrementing the complexity of the system gradually. Some times introducing a new object in the simulation changes the system logic, in that case the simulation-based process should start from the beginning.

	Achievements
As-fast-as-it-can simulation	Fast, Coarser perspective of the system, System Long-run behavior, Logic test, System goal achievement.
One real robot	Design consistency, Design efficiency, Design the interface with real objects
Two real robots	Exploring the dimensionality and heterogeneity, Robustness
More real robots	Resource management, Exploring the complexity of new objects and their common factors
Distributed convoy	Test the effect of the design in practice, Design reliability

Table 4.7: The intermediate results of incremental simulation based design

The overall design of the software here was based on supervised development and simulated test. The simulation was used not only to fill the gap of the to-be-expected objects but also as a feedback loop helping the designer to overcome the problems. The result of this step-wise development is a system along with its model being simulated. This code written for the simulated model can be directly applied to the real objects now (as we plan to do for the distributed team formation), and since a near perfect model is in hand, it can be used as a substitute of the system in building a bigger system. The model can also be consulted, if the real system is no longer available or hard to manage or control. The latter benefit can be used for example in emergency decision makings of an aircraft when an engine stops. Since the system runs the same code written for the model, the model can best describe the system, therefore it helps predicting the behavior of the system or even finding the defective part.

4.7 System-in-the-loop

The problem solving mechanism in robot-in-the-loop, as discussed in this thesis, is a test-bed for investigating the model continuity as a technology for iterative software test and development. A more general case is to look the problem as a scientific problem solving method for developing any complex system using modeling and simulation. The very first phase starts with modeling objects in an abstract way to test the overall logic of the problem from a coarser perspective. In another words to explore the problem space to find the conflicts as fast as possible using simulation. In each step, by introducing a new dimension to the problem, a finer resolution is investigated which leads to less abstraction of the model. The model is developed along with the system while keeping the logic and design goals.

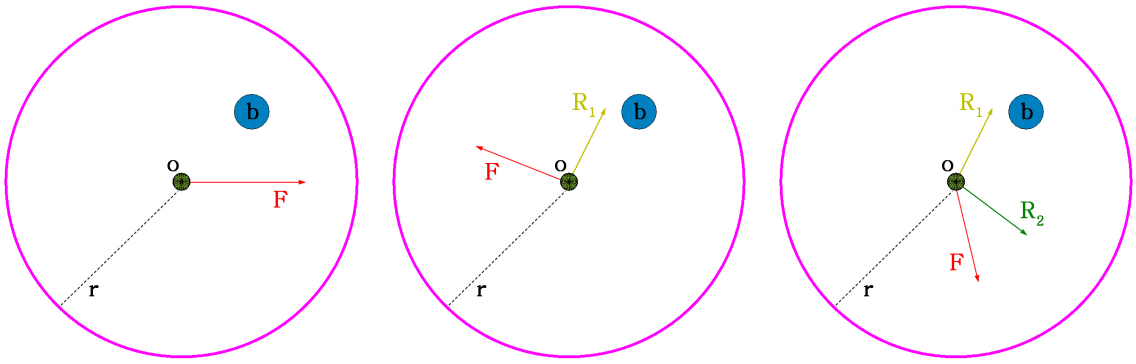

Figure 4.8: From left to right, a) First level, abstracted problem space b) Second level, adding one dimension c) Third level, adding second dimension

To visualize this technique, assume the objects as forces F which try to move a box placed in o to the specific distance of r. As shown in Figure 4.8(a), there are infinite solutions to the problem, which one solution is shown. Because of the abstraction of the objects, all forces are put in one dimension (assimilating the case of all identical

virtual robots). The barrier b is a logical conflict (as the deadlocks in robot problem), and can be found by exploring the problem space of polar coordinate system.

To further discover the problem, we apply another force in a new dimension R_1 (Fig. 4.8(b)) not inline with the first dimension. This case assimilates the step in which we added only one real robot to the convoy, when reality brings a new dimension to the problem with unchangeable force. By changing the object models (F) the search is again performed for a new solution, of course more limited than before. Going to the next phase, the heterogeneity of real objects adds new dimensions to the problem (shown by R_2 in Fig 4.8(c)) in the course of the problem solving, while further restricting the solution domain. Figure 4.9 illustrates the problem space after bringing a third real object. The forces and also the barrier (b) now have more dimensions in a way that the older abstract is the top view of the new problem space.

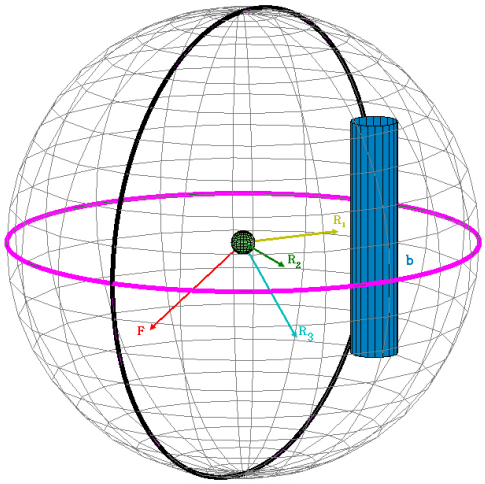

Figure 4.9: Bringing the third dimension

Unfortunately when bringing more and more dimensions however, they are not always independent from each other. The problem is dynamic in a sense that its

dimensions have some common factors. For example, some real objects may have a common limited source that is shared among them (like the bandwidth communication with real robots). There are also some unknown effects of dimensions on each other, requiring further investigating each dimension before moving to the next steps.

4.8 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 navigational 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.

In chapter 2, the developed system architecture (including hardware and software) is presented. The hardware setup includes the the overhead camera, wireless communication with the computer, a field (for robots to move) and robots. The several software units of the system and their interactions are also introduced. Most of these components are configured by an interactive GUI to see the result of different parameters on the system. The software, in the central team formation, is in charge of controlling and monitoring the team formation. In the distributed team formation however, the software would be used only for team formation monitoring, analyzing

and broadcasting the locations of robots to the distributed on-board software of the robots.

In chapter 3 the localization of the robots (based on image processing), which is actually another unit of the software architecture, is introduced. A circular pattern is used in the image processing to find the ID and heading of each robot. This pattern uses a color coding scheme which is then identified in the grabbed image of the field. The image processing algorithm, the effect of noise and the criteria to choose certain marker patterns (to place on top the robots) are discussed. The localization result is used as the sensory input of the behavior choice mechanism.

In chapter 4, we give the simulation results. The agents (virtual and real robots) are simulated (bringing one real robot at a time), to further characterize the behavior tasks and build better robot models. To improve the performance and solve some issues which are discovered during the simulations, the tasks are defined by a combination of simple navigational commands. In the modeling of the behavior network we do not take into account any specific parameter for individual robots, therefore dealing with the intrinsic heterogeneity of the real robots requires more robust models (which does not depend on the individual robots). The robotic collaboration is an example of the robot-in-the-loop simulation technique, but this technique can be extended to be used in other system developments as well.

As the future work, we will complete the second part of the project (which is the distributed team formation) to analyze the models in a team of all real robots each navigating based on its own embedded behavior network. Various possibilities exist for further study and use the current system in hand. The system is engineered so that the main components can be used in (or combined with) other applications (with

robotics as a tool) such as a behavior simulator for the robots (the behavior network is built interactively by a GUI connecting the inhibition links), to teach the concepts of robotics, behavior mechanism, DEVS modeling and simulation. Finally, the control aspects of the robotics (such as PID controllers), applying other multi robot missions rather than the team formation (such as the search and rescue missions), improving the image processing algorithm, fusing localization data with the infra red proximity distances of the robots, design for fault tolerance, and emergent behavior design (for example by eliminating the absolute order of the team formation) are other potential future researches.

Bibliography

[1] R.G. Menendez and J.E. Bernard. "Flight simulation in synthetic environments". *Proc. of Digital Avionics Systems Conferences (DASC)*, 1:2A5/1–2A5/6, 2000.

[2] E. Moritz and J. Meyer. "Interactive 3D protein structure visualization using virtual reality". *Proc. Bioinformatics and Bioengineering (BIBE)*, May 2004.

[3] Soha Maad and Saida Bouakaz. "From Virtual to Augmented Reality in Financial Trading: A CYBER-II Application". *European Conference on Combinatorial Optimization ECCO XVII*, 2004.

[4] 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)*, pages 90–98, 1997.

[5] 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)*, pages 759–764, 1997.

[6] 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)*, 1:293–297, 1999.

[7] 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)*, 2:1161–1166, 1999.

[8] E. Freund and J. Rossmann. "Projective virtual reality: bridging the gap between virtual reality and robotics". *IEEE Transactions on Robotics and Automation*, 15(3):411–422, 1999.

[9] X. Hu and N. Ganapathyand B.P. Zeigler. "Robots in the loop: supporting an incremental simulation-based design process". *IEEE International Conference on Systems, Man and Cybernetics*, 3:2013–2018, 2005.

[10] Bertrand P. Zeigler and Hessam S. Sarjoughian. *"Introduction to DEVS Modeling and Simulation with JAVA: Developing Component-Based Simulation Models"*. jan 2005.

[11] Bertrand P. Zeigler, Herbert Praehofer, and Tag Gon Kim. *Theory of Modeling and Simulation: Integrating Descete Event and Continious Complex Dynamic Systems*. Academic Press, 2 edition, 2000.

[12] X. Hu and D. H. Edwards. "Context-Dependent Control of Adaprive Behavior Selection". *Proc. Workshop on Bio-inspired Cooperative and Adaptive Behaviors in Robots, in co-operation with The Ninth International Conference on the Simulatio of Adaptive Behavior (SAB06)*, 2006.

[13] X. Hu. *A Simulation-Based Software Developement Methodology for Distributed Real-time Systems*. PhD thesis, University of Arizona, Department of Electrical and Computer Engineering, 2004.

[14] C.M. Krishna and K.G. Shin. *Real-time Systems*. McGraw-Hill, New York, 1997.

[15] 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*, 7:355–375, 1997.

[16] X.Hu, B. P. Zeigler, and J. Couretas. "DEVS-on-a-Chip Implementing DEVS in Real-time Java on a tiny Internet Interface for Scalable Factory Automation". *IEEE International Conference on Systems, Man, and Cybernetics*, October 2001.

[17] Bertrand P. Zeigler, Yoonkeon Moon, Doohwan Kim, and Jeong Geun Kim. "DEVS-C++ : A High Performance Modeling and Simulation Environment". *HICSS (1)*, 7:350–359, 1996.

[18] X. Hu, B. P. Zeigler, and S. Mittal. "Variable Structure in DEVS Component-Based Modeling and Simulation". *SIMULATION: Transactions of The Society for Modeling and Simulation International*, 81(2):91–102, 2005.

[19] X. Pan. *Computational Modeling of Human and Social Behaviors for Emergency Egress Analysis*. PhD thesis, Stanford University, Civil and Environmental Engineering Dept, 2006.

[20] Nigel Gilbert. "Agent-Based Social Simulation: Dealing with Complexity". *Centre for Research on Social Simulation, University of Surrey*, December 2004. Guildford, UK.

[21] Xiaolin Hu. "Context Dependent Adaptability in Crowd Behavior Simulation". *IEEE International Conference on Information Reuse and Integration (IRI 2006)*, 2006.

[22] Samuel H. Kenyon. "Behavioral software agents for real-time games". *IEEE Potentials*, 25(4):19–25, July 2006.

[23] J. K. Rosenblatt and D. W. Payton. "A Fine-Grained Alternative to the Subsumption Architecture for Mobile Robot Control". In *Proc of the IEEE Int. Conf. on Neural Networks*, volume 2, pages 317–324, Washington, DC, 1989. IEEE Press.

[24] James McLurkin. Stupid Robot Tricks: A Behavior-Based Distributed Algorithm Library for Programming Swarms of Robots. M.sc., M.I.T., May 2004.

[25] D. D Dudenhoeffer and M.P. Jones. "A Formation Behavior For Large-Scale Micro-Robot Force Deployment". *Proceedings of the 2000 Winter Simulation Conference (WSC '00)*, December 2000.

[26] Donald H. Edwards. "Mutual inhibition among neural command systems as a possible mechanism for behavioral choice in crayfish". *Journal of Neuroscience*, 11:1210–1223, 1991.

[27] X. Hu and D. H. Edwards. "BehaviorSim: A Simulation Environment to Study Animal Behavioral Choice Mechanisms". *Proc. of the 2005 DEVS Integrative M&S Symposium*, 2005. San Diego CA.

[28] Fredrik Linåker. *Unsupervised On-line Data Reduction for Memorisation and Learning in Mobile Robotics*. PhD thesis, University of Sheffield, Department of Computer Science, 2003.

[29] Charles Barrasso. Vision Based Monte Carlo Localization on the Khepera II. Master's thesis, Rochester Institute of Technology, Department of Computer Science, 2005.

[30] D. Fox, J. Ko, K. Konolige, B. Limketkai, D. Schulz, and B. Stewart. "Distributed Multi-robot Exploration and Mapping". *Proceedings of the IEEE*, 94(7):1325–1339, 2006.

[31] Pierre Bureau for K-Team S.A. *Khepra^2 programming manual*. K-Team, Ch. de Vuasset, CP 111, 1028 Prverenges,Switzerland, 0.2 edition, June 2002.

[32] K-Team S.A. *Khepra^2 user manual*. K-Team, Ch. de Vuasset, CP 111, 1028 Prverenges Switzerland, 1.1 edition, March 2002.

[33] Bryan Scotney, Sonya Coleman, and Dermot Kerr. "A Graph Theoretic Approach to Direct Processing of Sparse Unwarped Panoramic Images". *IEEE Internationl conference on Image Processing*, October 2006.

[34] H. Bastani, H. M.M. Sadeghi, A. Shariatmadari, E. Azarnasab, and M. Davarpanah-Jazi. "Design and Implementation of a Full Autonomous Mine Detecting Robot". *5th IFAC/EURON Symposium on Intelligent Autonomous Vehicles*, 2004.

[35] Wu Yiming and Liu Xiuwen. "Two-Stage Optimal Component Analysis". *IEEE Internationl conference on Image Processing*, October 2006.

[36] D. J. Bruemmer, D. D. Dudenhoeffer, and J. Marble. "Mixed-Initiative Remote Characterization Using a Distributed Team of Small Robots". *AAAI Mobile Robot Workshop*, 2001. Seattle, WA.

[37] James McLurkin and Jennifer Smith. "Distributed Algorithms for Dispersion in Indoor Environments using a Swarm of Autonomous Mobile Robots". *Distributed Autonomous Robotic Systems Conference*, 2004.

[38] P.M. Downes, M.J. Kwinn, and D.E. Brown. "Using agent-based modeling and human-in-the-loop simulation to analyze army acquisition programs". *Proc. of the 2004 Winter Simulation Conference*, 1, 2004.

[39] Parimal Kopardekar, V. Battiste, W. Johnson, R. Mogford, E. Palmer, N. Smith, J.F. D'Arcy, K. Helbing, P. Mafera, P. Lee, J. Mercer, T. Prevot, and S. Shelden. "Distributed air/ground traffic management: results of preliminary human-in-the-loop simulation". *The 22nd Digital Avionics Systems Conference*, 1:5D3–51–12, 2003.

[40] X. Hu and B. P. Zeigler. "Model Continuity in the Design of Dynamic Distributed Real-time Systems". *IEEE Transaction on Systems Man and Cybernetics*, pages 867–878, November 2005. Part A.

[41] X. Hu and B. P. Zeigler. "Model Continuity to Support Software Development for Distributed Robotic Systems: a Team Formation Example". *Journal of Intelligent and Robotic Systems, Theory and Application*, pages 71–87, January 2004.