

Situated Robot Design with Prioritized Constraints

Pınar Muyan-Özçelik and Alan K. Mackworth
Department of Computer Science
University of British Columbia
Vancouver, BC, Canada V6T 1Z4
{pmuyan,mack}@cs.ubc.ca

Abstract—The Constraint-Based Agent (CBA) framework with prioritized constraints is a simple and effective methodology for designing and building situated robots. This methodology can be seen as a formal development of the subsumption approach. It prevents ad hoc layering of behaviors and supports modular development of the system.

A situated robot called Ainia that repeatedly finds, tracks, chases, and kicks a ball is presented as an illustrative case study of this design methodology. Ainia is first modeled, simulated, and animated with the Constraint Nets in Java (CNJ) tool. Then, a prioritized constraint-based controller of the simulated Ainia is used to control the physical robot in the real world.

The results show that the behaviors of the physical robot satisfy the requirements specification. Hence, this study provides evidence that the formal CBA framework with prioritized constraints is an effective approach for synthesizing situated robot controllers. In addition, it supports the claim that CNJ is an effective tool for designing and building situated robots operating in the real world.

I. INTRODUCTION AND MOTIVATION

The Constraint-Based Agent (CBA) framework developed by Zhang and Mackworth [1] is a formal approach to building hybrid systems as situated agents. Situated agents follow the “situatedness” principle of Brooks [2]. There is a strong two-way coupling between the body of the agent and its environment. Situated agents are designed for particular tasks and environmental niches.

The CBA approach consists of several formal models which are used for the interleaved phases of dynamic system modeling, requirements specification (or “behavior specification”), control synthesis, and behavior verification. Prioritized constraints are specified and modeled in the requirements specification and control synthesis phases, respectively. Asimov’s three laws of robotics are examples of prioritized constraints [3], [4].

The CBA framework with prioritized constraints can be seen as a formal development of Albus’ model [5] and subsumption architecture [6]. Subsumption is a reactive architecture developed by Brooks which focuses on priority-based arbitration of task-achieving behaviors. The name subsumption arises from the design, where higher behaviors are added on top of lower behaviors by using priority-based arbitration. Hence, complex behaviors subsume simpler behaviors.

In subsumption, behaviors are often layered in an ad hoc way without using any formal specification or structured

rules of hierarchy. In addition the upper and lower layers in the architecture cannot be designed completely independently since the upper layers interfere with lower layers. Hence, modularity is not supported.

The CBA framework with prioritized constraints addresses these shortcomings of the subsumption architecture. The interaction hierarchy of the Constraint Nets (CN) modeling language and the formal requirements specification used in the CBA framework avoids the ad hoc layering of the behaviors in the system. In addition, the CN language supports modularity. In this language, modules can be modeled and debugged independently and then glued together.

By supporting modularity and using prioritized constraints, the CBA framework proposes a practical approach to control synthesis. To synthesize the controller, the prioritized constraint specification of the system and the constraint-solving behavioral modules for each of the specified constraints should be provided. Then, the controller of the system can be synthesized with connecting modules by the arbiters in the specified priority order.

Constraint Nets in Java (CNJ) developed by Song [7] is a visual programming environment for CN modeling, simulation and animation. Related tools that support visual programming for control system modeling and synthesis are Simulink [8], ControlShell [9], and MissionLab [10].

Previous studies employing the CBA approach have worked on a robot that can escape from mazes [11], a two handed robot that assembles objects [11], an elevator simulation which serves requests of the users in a building [12], soccer-playing robots [13], [14], and the like.

All of these systems proposed that the CBA approach is an effective and practical design framework. However, they utilized a limited version of the CBA framework, which did not have the prioritized constraints. Additionally, only two of these systems were specified and animated in CNJ: the dynamics of a soccer-playing car-like robot [7] and the dynamics of an elevator [15]. Moreover, these two systems were not used in the real world, after being modeled and simulated with CNJ.

With the intention of filling the voids mentioned above, the motivation for this study was to provide evidence that the CBA framework with prioritized constraints, using CNJ, is an effective approach to building situated robots in the real world.

As an illustrative case study of this design methodology, a situated robot called Ainia, that repeatedly finds, tracks, chases and kicks a red ball in the field was first designed and simulated in CNJ. After modeling the controller, the body, and the environment of Ainia as separate modules and creating an animation of the system under CNJ, the controller module was used unchanged to control a physical robot. The body and environment modules were replaced by the physical robot plant and the external real world, respectively.

II. THE CONSTRAINT-BASED AGENT FRAMEWORK

The CBA framework differs from other methodologies for hybrid systems. It proposes a formal, unitary, modular, and constraint-based approach based on the following four theses: 1) Hybrid systems should be modeled, specified and verified using formal methods. 2) The approach should have a unitary framework that supports both discrete and continuous time/domain structures. 3) The model should support hierarchy and modularity. 4) Agent controllers should be specified and designed as online constraint satisfiers.

The CBA framework proposes viewing constraints as relations among phase space variables and constraint satisfaction as a dynamic process of approaching the (possibly time-varying) solution set of the constraints persistently. Since task requirements, physical limitations and environmental restrictions can usually be specified as constraints, most robotic systems are constraint-based dynamic systems. Thus, the CBA approach to controller design, which advocates building robotic controllers as constraint solvers that perform online constraint satisfaction, is a dependable and scalable approach.

The approach consists of several formal models which are used for the interleaved phases of dynamic system modeling, requirements specification (or “behavior specification”), control synthesis and behavior verification. In the CBA framework, a CN formal model is used for modeling the dynamics of the plant and its environment and for control synthesis. Timed \forall -automata are used for requirements specification and a formal model checking method is used for behavior verification.

In CN, we generate a dynamic system model that represents the whole system as a set of components and their relations. The CN model can express the underlying structure of the system; however, it can not explicitly express the behaviors of the system. Behavior specification is used for this purpose. In timed \forall -automata, we can restrict requirements specification to constraint-based specification. This specification explicitly represents the desired behaviors (dynamic relationships between the robot and its environment) of the robotic system. Hence, in the CBA framework, a controller acts as a constraint solver to generate behaviors.

Building a robotic controller such that the behaviors of the robotics system meet the given requirements is referred to as control synthesis. In the CBA framework, requirements specification and the dynamics of plant and

its environment are needed beforehand to synthesize a correct controller. Behavior verification is used in parallel with control synthesis in order to build the correct robotic system. It makes sure that behaviors of the system generated by the constraint solving controller meet the specified constraint-based requirements.

In this paper, we focus on system modeling and control synthesis; the behaviour specification and verification phases are not explained in detail.

Constraint-based agents modeled in CN are built in three separate modules: the controller, body, and environment modules (Figure 1). The controller and body modules are coupled together to constitute the agent. Likewise, the agent and environment modules are coupled together to constitute the agent’s system. This structure allows CN to model agents that are situated in their environments.

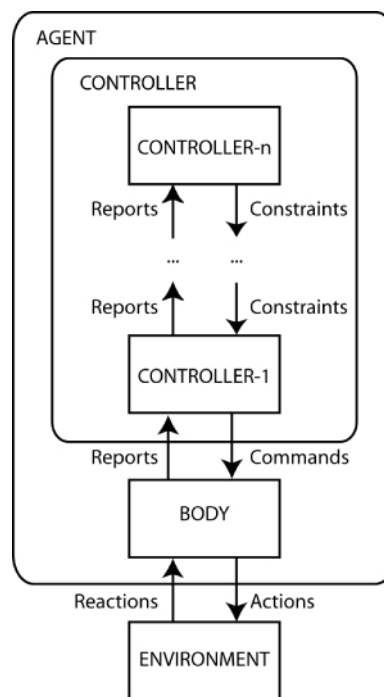


Fig. 1. The structure of a constraint-based agent system

The body is the direct interface of the agent to its environment. It executes actions in the outside world, senses the changes in the environment and reports these changes to the controller. Given the current states of the environment and the body, the controller computes and sends proper commands to the body that would satisfy the constraints.

In CN, the controller can be constructed with several levels in a hierarchy. In this hierarchy, each level solves constraints that are set by the level above and produces the constraints to be sent to the lower level. Typically, the lower levels are reactive and upper levels are deliberative. If the situated agent needs to do dynamic planning (e.g. it has more than one goal to achieve or it has to change its behaviors by predicting the future) the controller should have a deliberative level. Ainia does not have to

do dynamic planning. Thus, its controller has only one level. Within this level the constraint-solving behavioral modules are layered in a structured way. The layering of the constraints corresponds to their fixed priority ordering; it implicitly represents the robot's static plan.

Syntactically, CN is a triple $CN = \langle Lc, Td, Cn \rangle$ where Lc is a finite set of locations, Td is a finite set of transductions, and Cn is a set of connections. Locations are variables, each of which is associated with a data type (e.g. integer, boolean, and the like). They denote traces, which characterizes the sequence of values of variables over time. Transductions act like functions; they represent casual mappings from input traces to output traces, operating on global reference time or activated by external events. A transduction has a set of input ports and an output port, which are also associated with data types. Connections, on the other hand, represent the interaction structure of the system by connecting locations with ports of transductions.

Semantically, the CN model is the solution of a set of equations. The solution is a temporal trace of values resulting from the transductions over time. Hence, CN provides an online model of computation instead of an offline model in which a solution is a function of pre-given inputs. In CN, equations are realized by locations (variables) and transductions (functions). Possible input/output traces produced by the system define behaviors of the dynamic system. The overall semantics of a system can be obtained hierarchically from the semantics of its subcomponents and internal connections.

Graphically, a CN is illustrated by a bipartite graph where locations are depicted by circles, transductions by rectangles, and connections by arcs. Modules are depicted by boxes with round corners. They encapsulate locations, transductions, and connections. They may also encapsulate other submodules. Modules have a set of exported locations as their interfaces.

III. THE CONSTRAINT NETS IN JAVA TOOL

CNJ is a visual programming environment for CN modeling, simulation and animation. Since CN has inheritance hierarchies composed of modules and graphical nodes such as locations, transductions and connections, it is an ideal model for visual programming. To provide a set of building blocks for CN modeling, CNJ uses JavaBeans technology that supports software components. The following nodes in CN are realized by JavaBeans: locations, transductions, clocks, and modules. In addition, the connection node of CN is implemented by Java's event mechanism.

CNJ uses Java's Swing capabilities for its Graphical User Interface (GUI) design. It has two main windows, namely *CNFrame* and *BeanPropertySheet*, as shown in Figure 2.

The *CNFrame* window is the bigger one; it is used for designing and drawing a CN. It includes *DrawPane* and *ToolPane* windows. In CN, modules are located in a hierarchy to compose the system. To support as many modules as possible, CNJ uses the Multiple Document Interface approach and displays each module in a child

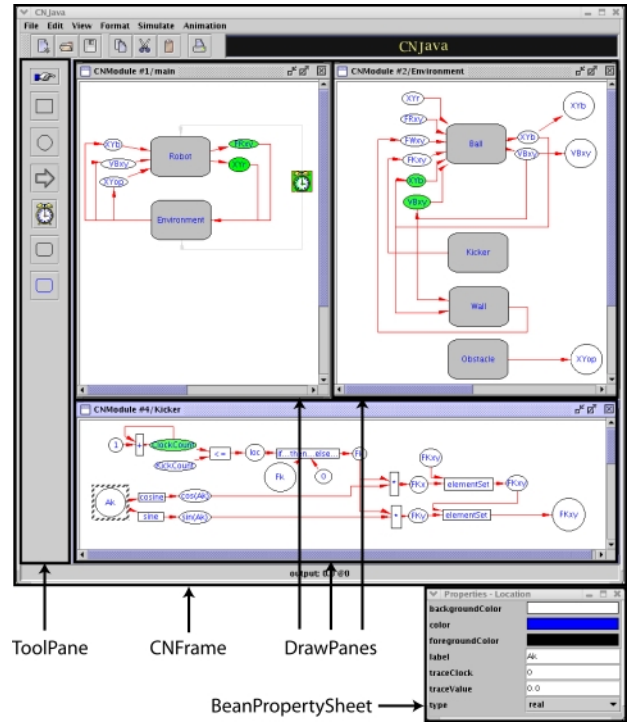


Fig. 2. The GUI of CNJ

DrawPane window. To design a CN, the designer needs to choose an element (e.g. location, transduction) from the *ToolPane* and then drag-and-drop it to the selected *DrawPane*. The locations and ports of transductions are wired by clicking the *connection* element of *ToolPane*. Therefore, CNJ allows designers to program visually and “draw” a CN model instead of writing code for it.

In the GUI of CNJ, the *BeanPropertySheet* window, the smaller window at the bottom right, is used to display and customize corresponding properties of the focused node in the selected *DrawPane* window.

Before the model can be simulated in CNJ, it goes through a compiling step. This step is able to uncover some errors in the design of CN model, such as uninitialized values, data type incompatibility between the connected nodes, ill-defined transductions, and so on.

In CNJ, simulation of the system is realized by Java's event mechanism and driven by clock tick events. Clock is a special kind of transduction with no input locations; it can be drawn by selecting the *clock* element of *ToolPane*. During the simulation the designer can watch the results (traces that define the behavior of the system) from the output locations or can link these results to an animation window in order to analyze the overall behavior of the system. In CNJ, the animation window can be implemented with the Java 3D API. A CN model is stored in the Constraint Net Markup Language (CNML), an XML-based interchange file format.

IV. AINIA AND ITS PRIORITIZED CONSTRAINTS

Ainia is a situated robot which has the task of repeatedly finding, tracking, chasing and kicking a red ball in the field.

Ainia is quick to react the dynamic changes in the world, so she is named after an Amazon warrior whose name means “swiftness”.

The body of the physical Ainia is constructed of three main components as shown in Figure 3:

1) B14 Mobile Robot manufactured by Real World Interface, RWI, Inc. which includes 850Mhz Intel Pentium III onboard processor with 256 MB memory and Red Hat Linux 7.2 (Enigma) operating system; 10 Mbit Compaq wireless ethernet modem and Compaq range extender antenna that allow communication to a remote computer; and motion and power components such as batteries, wheels and base translation/rotation motors.

2) Digiclops trinocular stereo vision color camera manufactured by Point Grey Research Inc. which allows visual sensing of the outside world. It has three identical wide angle cameras, arranged in an L shape. However, since stereo and depth data are not needed to accomplish Ainia’s task, only the image taken from the bottom right camera is used by the robot software.

3) PTU-46-17.5 pan/tilt unit manufactured by Directed Perceptions Inc. which is mounted on top of the B14 Mobile Robot. It has 317° pan range and 77° tilt range. The Digiclops camera is mounted on top this component. Therefore, the pan/tilt unit enhances the visual sensing of the environment by allowing the camera to have more degrees of freedom, complementing the motions of the B14 Mobile Robot.

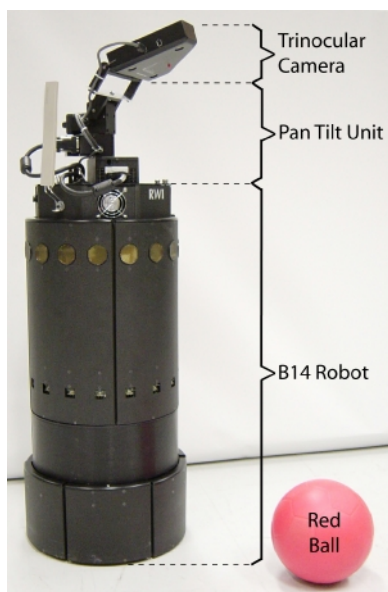


Fig. 3. Ainia and the red ball

In order to accomplish its task, Ainia should first find the ball by using pan/tilt motions and base rotation motion if necessary. Consequently, the ball should be in the camera image. This constraint is denoted as I (short for “BallInImage”) in the rest of the paper. After it finds the ball, it should center the ball in the image, by panning and tilting in the ball direction. This constraint is denoted as C

(short for “BallInCenter”) in the rest of the paper. Then, it should turn its base towards the pan direction while keeping the I and C constraints satisfied. This constraint is denoted as H (short for “BaseHeadingPan”) in the rest of the paper. Finally, as soon as the pan and the base are aligned, it should move forward to kick the ball. This constraint is denoted as A (short for “RobotAtBall”) in the rest of the paper. Note that, since the robot can move forward and backward immediately, but it cannot move sideways without turning its base first, Ainia needs to satisfy the H constraint before it can solve the A constraint.

Thus, Ainia’s behavior specification involves four prioritized constraints which are ordered as follows: $I > C > H > A$. Formal specification of the prioritized constraints can be done by timed \forall -automata as presented in [16]. Note that in this specification $X > Y$ denotes X has a higher priority than Y , and thus the behaviors of the systems should solve the X constraint before it can satisfy the Y constraint. However, the constraints are dynamic and may not stay solved. Hence, the behaviors of the system may have to re-solve the higher priority constraints as needed.

Ainia’s goal can be defined as always eventually kicking a red ball in the field. However, the previous requirements specification does not explicitly specify a behavior that would solve this goal. Instead it specifies simple behaviors that when combined produce an “emergent” behavior that would satisfy the goal.

Instead of using its base rotation motors, Ainia primarily uses its pan and tilt motors in order to find and center the ball. This most closely mimics the mechanism of the human visual system, since humans turn their heads to look for objects, instead of turning their entire body. In addition, since the speeds of the pan and tilt motors are higher than the speed of the base rotation motor, primary usage of the pan and tilt motors allows the robot to explore different parts of the world more efficiently.

V. SIMULATED AINIA

CNJ allows us to simulate and animate Ainia’s robotic system. This provides an important advantage of testing and developing our system without constructing a physical base for it. During the design phase, working with simulations is less expensive than working with real systems and it dramatically speeds up the construction of the correct controller.

The simulated Ainia’s robotic system can be modeled under CNJ by coupling the *Environment* module and the *Robot* module as shown in Figure 4. Note that details of the modules and descriptions of the locations used in the simulated Ainia’s CN can be found in [16].

The *Environment* module encapsulates the *Ball*, *Kicker*, *Wall*, and *Obstacle* modules. These submodules model the dynamics of the ball as well as interactions of the kicker and the ball with the four walls (which surround the environment) and the obstacles (which can be created by the CNJ user). On the other hand, the *Robot* module encapsulates coupling of the *Controller* and *Body* modules as shown in Figure 5.

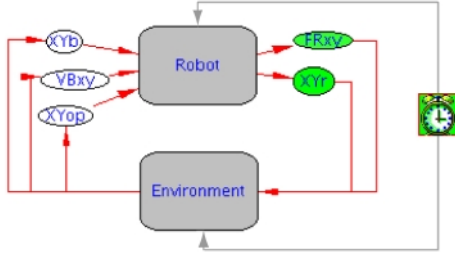


Fig. 4. Simulated Ainia's *Main* module

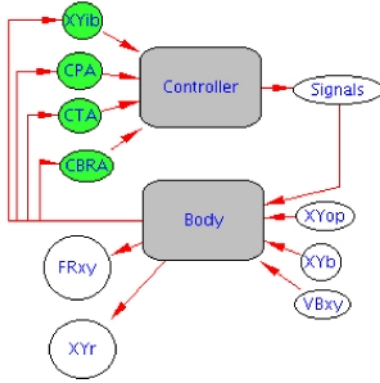


Fig. 5. Simulated Ainia's *Robot* module

The *Body* module encapsulates the *Pan*, *Tilt*, *BaseRotation*, *BaseTranslation*, *Camera*, and *BaseBump* modules. These submodules model the dynamics of the body parts of the physical robot, i.e. pan and tilt motors, base rotation and translation motors, camera, and bump sensors.

After the prioritized constraint-based behavior specification ($I > C > H > A$) and the dynamics of the environment and body (*Environment* and *Body* modules) are provided, the controller of Ainia can then be synthesized. Behavior verification can be used in parallel with control synthesis to build the controller. However, we focus on control synthesis in this paper.

To synthesize the controller, first the *BallInImage* module which solves the *I* constraint, the *BallInCenter* module which solves the *C* constraint, the *BaseHeadingPan* module which solves the *H* constraint, and the *RobotAtBall* module which solves the *A* constraint are modeled. Then, given these constraint-solving modules and prioritized constraint specification, the *Controller* module is synthesized with connecting modules in the specified priority order using the arbiters as shown in Figure 6.

Submodules under the *Controller* module work in parallel and compete for the control of the motors. The arbiters decide which commands of which modules to send to the motors.

An arbiter is a transduction that takes two input vectors (e.g. outputs of two constraint-solving modules, *Signals_i* and *Signals_c* in Figure 6) and calculates an output vector. All the vectors have the same structure. The first

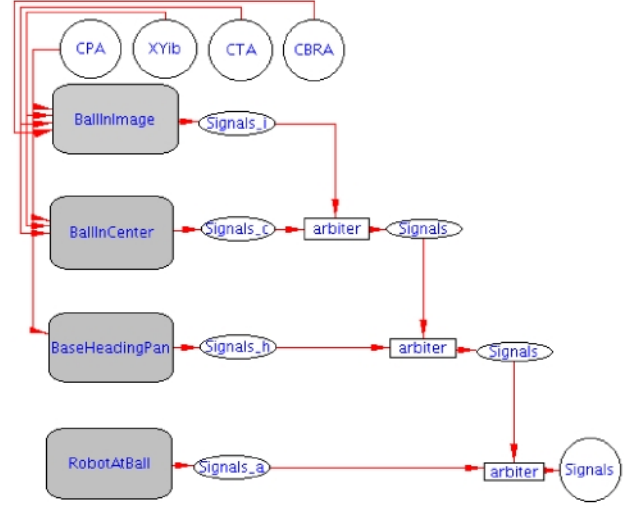


Fig. 6. Simulated Ainia's *Controller* module

element of these vectors refer to the satisfaction state of the related constraint(s). The rest of the elements refer to the commands to send to the motors. If the constraint-solving module does not control a specific motor it sends a *DoesntControl* signal to that motor. The arbiter calculates proper commands to be sent to all the motors. If the higher priority constraints are satisfied and the lower priority constraint controls the related motor, the command of the lower order constraint is sent to this motor. Otherwise, the command calculated by the higher priority constraint is used.

The *BallInImage* and *BallInCenter* behaviors control all the motors (i.e. the pan, tilt, base rotation, and base translation). The *BaseHeadingPan* and *RobotAtBall* behaviors do not control the pan and tilt motors. In addition, the *RobotAtBall* behavior does not control the base rotation motor. Hence, as long as the *I*, *C*, and *H* constraints are satisfied, the *BallInCenter* behavior can control the pan and tilt motors, the *BaseHeadingPan* behavior can control the base rotation motor, and the *RobotAtBall* behavior can control the base translation motor all at the same time. Note that, all behaviors except the *RobotAtBall* behavior command the base translation motor to stop.

The representation of the CN encapsulated in the *BaseHeadingPan* module is provided in Figure 7 as an example of a CNJ module.

In order to solve the *H* constraint the base motor should be heading towards the pan direction. Hence, the current pan angle (*CPA*) value should be very close to 0° . Consequently, the satisfaction signal of the *BaseHeadingPan* module can be calculated as follows:

$$Satisfy_h = \begin{cases} true & \text{if } |CPA| < TrHl.h \\ false & \text{otherwise} \end{cases}$$

where, *TrHl.h* is the threshold angle for checking whether *CPA* is close to 0° .

The *BaseHeadingPan* module commands the base rotation motor to turn towards the pan direction by setting

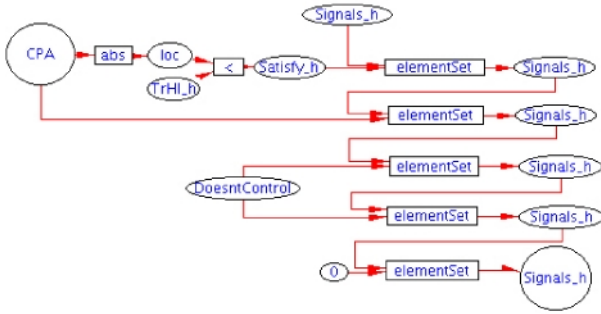


Fig. 7. Simulated Ainia's *BaseHeadingPan* module

the related element in the output vector to *CPA* value. It does not control the pan and tilt motors and sets the related elements to *DoesntControl* value. Finally, it commands the translation motor to stop by setting the related element to 0 speed value.

The animation window of the simulated Ainia is depicted in Figure 8. Notice that this figure provides a bird's-eye view of the robotic system.

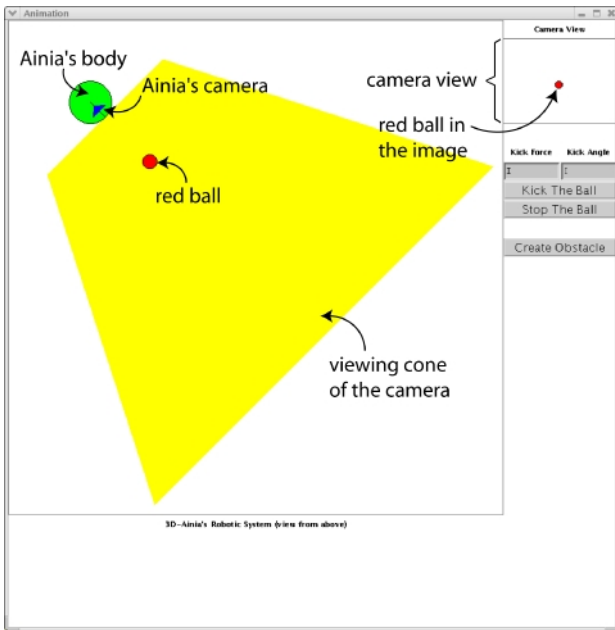


Fig. 8. The animation window of simulated Ainia

The animation of Ainia runs separately from its simulation. When an animation is turned on by the user from the *Animation* menu of *CNFrame*, a new thread is started. This thread creates an animation window which displays the behavior of the system by showing the dynamics of the robot body, the dynamics of the ball, and the view of the camera.

In order to paint the animation window, the thread retrieves current values of related locations from the simulation. Hence, whenever the animation window is repainted, new values of the locations calculated in the simulation are used and the changes in the system behavior are displayed

immediately.

VI. PHYSICAL AINIA

The animation of the simulated Ainia shows that the robot repeatedly finds, tracks, chases and kicks a ball on the field. Hence, the overall behavior of the system satisfies the requirements specification. This provides evidence that the controller synthesized in the simulation should function properly in the real world. Therefore to construct the physical Ainia, the *Controller* module of the simulated Ainia is used unchanged. The *Environment* and *Body* modules of the simulated Ainia are not used in the construction of the physical Ainia, since they are replaced with the real world and the physical robot body, respectively.

To control the physical robot, the *Controller* module should communicate with parts of the physical body through the software components which run on the robot and interface with the hardware. Since CNJ involves a GUI and requires an intensive use of memory and CPU, running this tool on the robot's onboard processor would not be an effective approach. Instead, the CNJ tool that runs the *Controller* module is installed on a remote computer and is connected to the robot, via wireless networking. Through this wireless connection, the remote computer and the robot establish sockets and send data to each other.

The software architecture of the physical Ainia consists of a CN that includes the simulated Ainia's *Controller* module, a communication coordinator developed for the physical Ainia and hardware interfaces of Human Oriented Messenger Robot, HOMER [17]. It utilizes the following hardware interfaces of HOMER: *RobotServer*, *ImageServer*, and *PTU Server*. These software components interface with the B14 Mobile Robot, camera, and pan/tilt unit, respectively. In this architecture the controller run on a remote computer and communication coordinator and hardware interfaces developed in C++ run on the robot.

The communication coordinator acts as a command executor and a report collector. It receives motion commands from the *Controller* module and sends them to hardware interfaces. In addition it retrieves reports from hardware interfaces and sends them to the *Controller* module. It communicates with hardware interfaces through the shared memory mechanism and it connects with the *Controller* module through sockets.

The communication coordinator also creates a GUI in order to display what the camera sees and to allow users to interact with the program through this displayed image. The user can interact with the program by choosing the type of the camera image to be displayed. The user can switch between a Color Image; a Red Image which shows red components as white and the rest as black; and a Ball Image, which shows a red component, that is most similar to a ball as white and the rest as black.

In order to find the ball-like component in the image, the communication coordinator applies a median filter to Red Image to reduce the noise. Then, it does sequential connected component analysis [18] on the resulting image. Components that have fewer than a certain number of

pixels are ignored. For all the components that are not ignored, compactness measure $\frac{P^2}{A}$ (where, P = Perimeter of the component and A = Area of the component) values are calculated. The component with the smallest value is selected as the ball-like component. If the shape of the component is close to a circle the above value would be minimized. The ball has a circle-like shape in the image. Therefore, it is successfully selected as the ball-like component by the communication coordinator.

VII. RESULTS

Experiments in the real world show that behaviors of Ainia satisfies the constraints specified in the system. Together with the dynamics of the body and the environment, the controller acts as a constraint solver and the robot always eventually kicks the ball with determination.

Behaviors of the physical Ainia are documented in the companion demonstration video. From this video, it can be seen that Ainia finds and tracks the moving ball and kicks it repeatedly by quickly responding to the dynamic changes in the world. In this paper photographs of the demonstration run are presented.

When there is no red object, and thus no ball, in the environment, the *BallInImage* behavior in the controller takes control of the motors and the robot goes into a loop searching for the ball. The state sequence of this search cycle is documented in Figure 9. As can be seen from this sequence, the *BallInImage* behavior commands the pan, tilt and base rotation motors and scans the floor surrounding the robot. When the red ball and a red book are placed on the floor Ainia successfully identifies the circular component as the ball, instead of the polygonal book. This stage is shown in Figure 10. In this figure an external view of Ainia’s robotic system is provided. Also, the corresponding camera view with three different image types (i.e. Color Image, Red Image, and Ball Image) are presented.

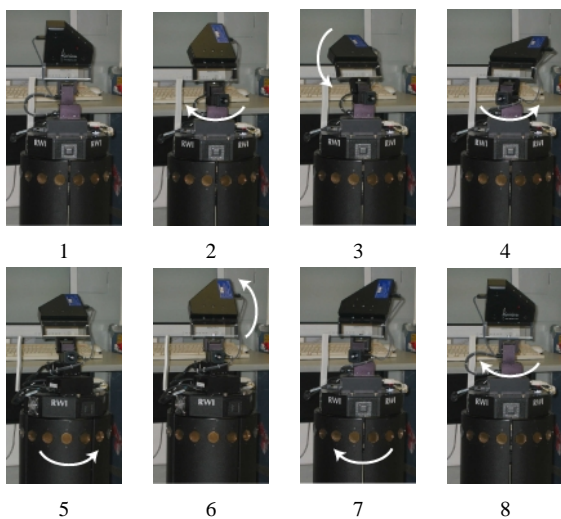


Fig. 9. States of the *BallInImage* behavior

At this stage, since the ball is in the image the I constraint is satisfied, the *BallInCenter* behavior takes

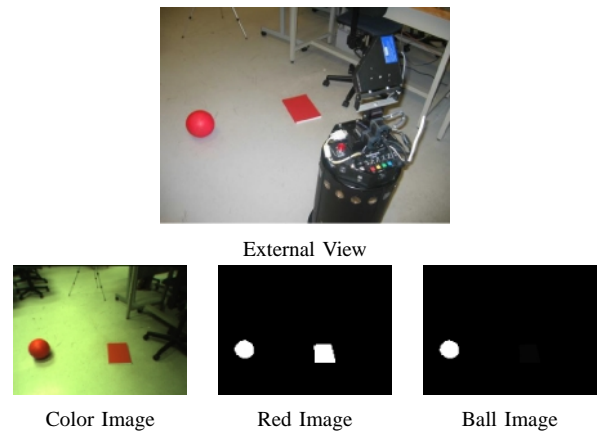


Fig. 10. Ainia identifying the ball

the control of the motors and centers the ball in the image. After the C constraint is satisfied, the *BaseHeadingPan* behavior becomes active and solves the H constraint by turning the base towards the pan direction. Then, the *RobotAtBall* behavior gets activated and the robot moves forward towards the ball (Stage1 in Figure 11). At this stage of the demonstration run, a human kicker kicks the ball (Stage2 in Figure 11) while the robot is moving forward. Since the C constraint becomes unsatisfied the robot is stopped by the *BallInCenter* behavior. This behavior once again takes the control of the motors and centers the ball in the image (Stage3 in Figure 11). Next, the *BaseHeadingPan* behavior becomes active again and solves the H constraint. Finally, the *RobotAtBall* behavior gets activated once more and the robot moves forward and kicks the ball (Stage4 in Figure 11).

Note that the *BallInCenter* and *BaseHeadingPan* behaviors take turns in controlling the motors until the H constraint is solved. This is because when the ball is centered in the image, the *BaseHeadingPan* behavior takes over the control of the motors and turns the base towards the pan direction. When the base is turned, the pan motor attached to the top of the base also turns and the ball becomes off-centered. In this case, the *BallInCenter* behavior takes over the control of the motors and once more it centers the ball in the image. Then, the *BaseHeadingPan* behavior again takes over the control and so on.

Ainia has some limitations. For instance, it is not always safe in the environment. By being unsafe we mean that it might bump into an object on the field and trip over. This limitation can be overcome by giving the highest priority to the safety constraint in the requirements specification and adding a safety module that solves this constraint on top of the *BallInImage* module in the controller. In addition, there are some situations in which Ainia does not perform very well. If the ball is going too fast or it is located too far away from the robot or there are some other red circle-like objects in the environment other than the ball, Ainia might not be able to achieve its goal.

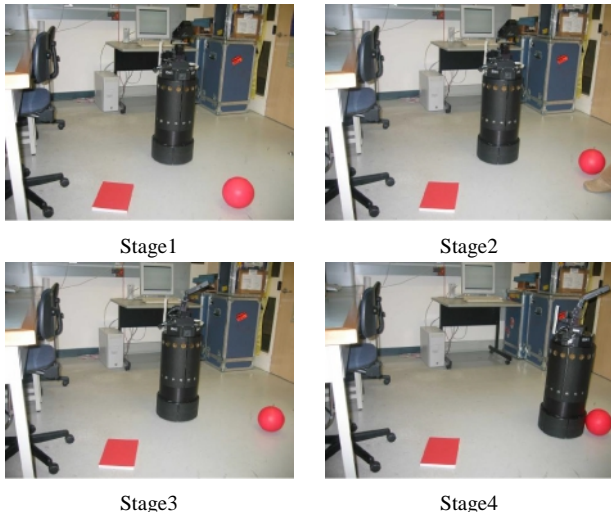


Fig. 11. Ainia tracking and chasing the ball

VIII. CONCLUSION

Results in the real world show that the overall behavior of the physical Ainia satisfies the constraint-based requirements specification. If the system limitations are not violated, the robot always eventually kicks the ball on the field.

These results provide evidence that the CBA approach with prioritized constraints is an effective framework for situated robot construction. The effectiveness of this method demonstrate that the usefulness of the subsumption approach can be enhanced by embedding it in the formal CBA framework. It also demonstrates that controllers for situated robots can be synthesized practically using this approach.

In addition, exactly the same controller was used for both the simulated and the physical Ainia. No changes were demanded in the controller while moving from the simulation to the real world. This supports the claim that CNJ is an effective real-time tool for designing and building situated robots, which operate in the real world. Hence, the CNJ tool has been proposed as a reliable and scalable tool for future research on the CBA framework.

There are many opportunities for improving Ainia. Ideas for future work include the implementation of a safety module in the controller to ensure that robot would always be safe in the environment. In addition, dynamic prioritization of the constraints in the controller can be investigated as future work. Static prioritization as used in Ainia, would not be adequate for designing and building more challenging situated robots. These systems would require dynamic planning and thus a deliberative level in the controller. For instance, a situated robot that can play soccer would have more than one goal to achieve (e.g. defending a goal as well as kicking the ball) which would require dynamic planning. Likewise, making Ainia track the ball more intelligently by predicting the future (e.g. beyond just using a Kalman Filter [19]) would also require a deliberative level. Hence, to allow for greater scalability of the approach, we could

implement dynamically prioritized constraints by adding a deliberative level onto the controller.

ACKNOWLEDGMENT

This research was funded by NSERC and Precarn/IRIS under the Robot PARTners project. Alan K. Mackworth holds a Canada Research Chair. We would like to thank Ying Zhang and Fenguang Song for their contributions to the development of the methods used in this study.

REFERENCES

- [1] A. K. Mackworth and Y. Zhang, "A formal approach to agent design: An overview of constraint-based agents," *Constraints*, vol. 8, no. 3, pp. 229–242, 2003.
- [2] R. A. Brooks, "Intelligence without reason," in *Proc. of 12th International Joint Conference on Artificial Intelligence - IJCAI-91, Sydney, Australia*, J. Mylopoulos and R. Reiter, Eds. San Mateo, CA: Morgan Kaufmann, 1991, pp. 569–595.
- [3] I. Asimov, *I, Robot*. Gnome Press, 1950.
- [4] D. Weld and O. Etzioni, "The first law of robotics (a call to arms)," in *Proc. of the 12th National Conference on Artificial Intelligence - AAAI 1994, Seattle, WA*. AAAI Press, 1994, pp. 1042–1047.
- [5] J. S. Albus, *Brains, Behavior and Robotics*. BYTE Books, McGraw-Hill, 1981.
- [6] R. Brooks, "A robust layered control system for a mobile robot," *IEEE Journal of Robotics and Automation*, vol. RA-2, no. 1, pp. 14–23, 1986.
- [7] F. Song and A. K. Mackworth, "CNJ: A visual programming environment for constraint nets," in *Proc. of AI, Simulation and Planning in High Autonomy Systems - AIS 2002, Lisbon, Portugal*, 2002, pp. 131–135.
- [8] *Simulink: User's Guide*, The MathWorks Inc., Natick, MA, 1992.
- [9] S. A. Schneider, V. W. Chen, and G. Pardo-Castellote, "The Control-Shell component-based real-time programming system," in *Proc. of IEEE International Conference on Robotics and Automation*, 1995, pp. 2381–2388.
- [10] D. C. MacKenzie, R. C. Arkin, and J. M. Cameron, "Multiagent mission specification and execution," in *Autonomous Robots*, 1997, vol. 4, no. 1, pp. 29–52.
- [11] Y. Zhang and A. K. Mackworth, "Will the robot do the right thing?" in *Proc. of Canadian Artificial Intelligence Conference - AI 1994, Banff, Alberta*, 1994, pp. 255–262.
- [12] —, "Modeling and analysis of hybrid systems: An elevator case study," in *Logical Foundations for Cognitive Agents*, H. Levesque and F. Pirri, Eds. Springer Verlag, 1999, pp. 370–396.
- [13] —, "A constraint based controller for soccer playing robots," in *Proc. of Int. Conf. on Intelligent Robots and Systems - IROS 1998, Victoria, BC*, 1998, pp. 1290–1295.
- [14] J. D. Montgomery and A. K. Mackworth, "Adaptive synchronization for a RoboCup agent," in *Proc. RoboCup-2002: robot soccer world cup VI, Fukuoka, Japan*, ser. LNAI, G. A. Kaminka, P. U. Lima, and R. Rojas, Eds. Springer Verlag, 2003, vol. 2752, pp. 135–149.
- [15] F. Song, "CNJ: A visual programming environment for constraint nets," Master's thesis, Department of Computer Science, University of British Columbia, Vancouver, 2002.
- [16] P. Muyan-Özçelik, "Prioritized constraints in the design of a situated robot," Master's thesis, Department of Computer Science, University of British Columbia, Vancouver, 2004.
- [17] P. Elinas, J. Hoey, and J. J. Little, "Human oriented messenger robot," in *Proc. of AAAI Spring Symposium on Human Interaction with Autonomous Systems in Complex Environments, Stanford, CA*, 2003, pp. 45–51.
- [18] R. Jain, R. Kasturi, and B. Schunck, *Machine Vision*. MIT Press and McGraw-Hill, 1995.
- [19] R. E. Kalman, "A new approach to linear filtering and prediction problems," *Transactions of the ASME—Journal of Basic Engineering*, vol. 82, no. Series D, pp. 35–45, 1960.