

Virtual Reality Simulation of Humanoid Robots

Yingheng Zhou and Ben Choi
Computer Science
Louisiana Tech University, LA 71272, USA
pro@BenChoi.org

Abstract - We developed a virtual reality system to simulate humanoid robots. Currently most humanoid simulation systems require manual manipulation of body parts or require capturing movements enacted by a person. We aim to describe humanoid motions using high-level language and created a humanoid motion description language. We defined new motion primitives and new syntax that allows programmers to easily describe complex humanoid motions. To make the motion specification possible, we defined a humanoid framework that models humanoid robots by specifying their capacities and limitations. Furthermore, we developed a simulation system that can execute the humanoid motion description language and can automatically handle conflicting motion statements and ensure that the described motions are within the limitations of humanoid robots. Our simulation results show that the proposed system has great future potentials.

1. Introduction

Virtual reality simulation of humanoids has been an important subject due to its wide range of applications such as motion picture production and robotics. To achieve realistic motions, programmers currently must be highly experienced to define all control factors for the movements of individual body parts on each time instance. Since it is far too difficult to control the detail moments of each individual body parts, currently motion capture systems are widely used to record motions enacted by a person and then to use the captured motions to animate humanoids.

In this paper, we attempted a high-level and goal-oriented approach. As shown in Figure 1, there are two extremes on the approaches to humanoid simulations, the goal-oriented and the motion-oriented. The goal-oriented approach specifies the objectives using high-level languages, that are easy to write, and that use Artificial Intelligent methods to generate detailed motions, but that are much more difficult in terms of building the system. On the other hand, the motion-oriented approach can specify detail motion using degree of freedom (DOF) of each joint. However, it is more complicated to write such specifications and programmers need to provide all the details, but such a system is earlier to be built.

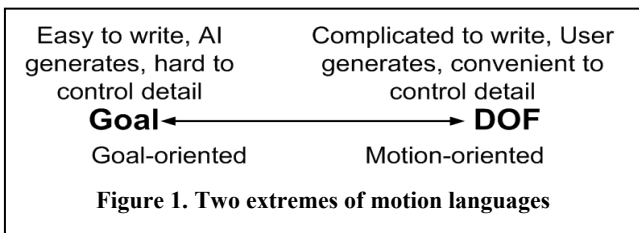
In this paper we present a new high-level goal-oriented language named Cybele. The language is designed for humanoid motion description [3] and is independent of

virtual reality simulation systems. Our new language provides simple syntax for expressing parallel and sequential processing as well as the combination of these two. We also introduce syntax for expressing motions. In particular, to solve the motion mixing problem in parallel and complex blocks, we present an approach to synthesizing motions with combination of partial states. We define key poses as states or partial states and create primitives between two key states. We also extract a relatively small set of parameterized motions from infinite complex motion sequences and make them reusable as primitive motions. Our simulator interprets the programs, breaks down motions into primitives with time, scope, and priority. Final motion sequences are then generated using a synchronization approach.

To achieve constraints satisfaction, we define the scope and priority for each joint in the humanoid. This system checks constraints on the joints affected by the motion and determines which motion is in conflict with some others. In addition, to create a multi-platform system, we implement a prototype system with two parts following the same paradigm as the Java virtual machine. The first part interprets the program and generates motion sequences. The second part translates motion primitives to interface with systems such as virtual reality simulation systems, animation systems, or robotic systems [9][6][15].

Our testing results show that our simulation system provides an effective and flexible way to create humanoid motion, and there is enormous potential for the future development.

The remaining of this paper is organized as follows. Section 2 outlines related work. Section 3 defines the new motion description language called Cybele. Section 4 describes our humanoid motion framework. Section 5 provides the implementation details of our humanoid



simulation system. And, Section 6 gives the conclusion and outlines the future work.

2. Related Work

This section outlines related work on the high-level language approaches to humanoid simulation. Several systems will be discussed, which including, Poser Python, VRML (Virtual Reality Modeling Language), Improv, STEP, and others. It focuses on high-level language approaches to humanoid simulation and omits other general concurrent languages such as OCCAM.

Poser Python [13][14] is an implementation of the Python interpreter that includes many commands that have been extended to recognize and execute commands not included in the standard Python language. Poser Python script language is a language combination that uses syntax and basic logic of Python and special commands tailored especially for Poser scene, manipulate them, and finally send them back to Poser. The language-controlled animation is a significant advantage of Poser-Python system.

VRML (Virtual Reality Modeling Language) is a scene-description language used widely on the internet. VRML uses TimeSensor to initiate and synchronize all the motions in the scene. It is possible that asynchronously generated events arrive at the identical time as one or more sensor-generated event. In such cases, all events generated are part of the same initial event cascade, and each event has the same timestamp. Based on this mechanism, VRML is quite suitable to visual presentations with user interactions. However, there is no direct way to describe complex motions with time overlapping.

Improv [11] is a system for the creation of real-time behavior based on animated actors. Improv consists of two subsystems. The first subsystem is an animation engine that uses procedural techniques to enable authors to create layered, continuous, non-repetitive motions and smooth transitions between them. The system uses an English-style scripting language so that creative experts who are not primarily programmers can create powerful interactive applications.

STEP [7] is a distributed logic program being used to define and control the hand gestures of embodied agents in virtual worlds. It works as an interface between the constructs of logic programming and the humanoid model defined in VRML. Using the STEP framework, different gesture dictionaries can be defined, and variants of a hand gesture, according to dynamically changing factors, can be generated on the fly. STEP also has two kinds of control logic definitions. One is parallel and the other is

sequential. But different kinds of program blocks cannot be mixed together and must be executed one by one.

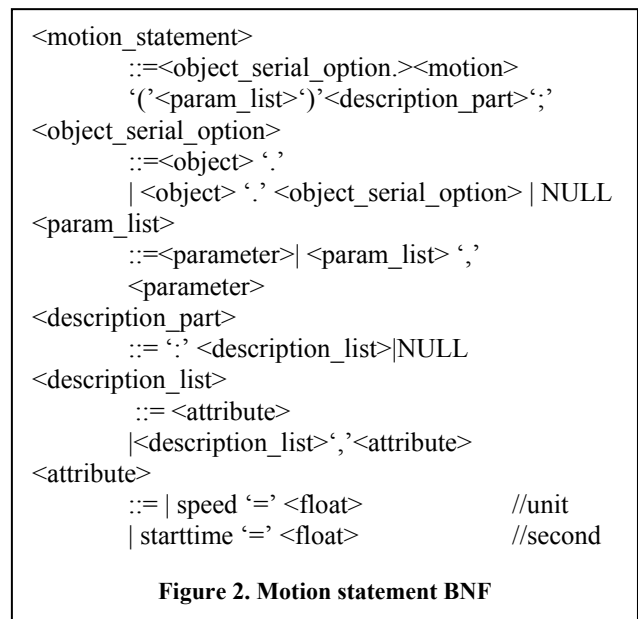
There are also some other systems for animations like Alice [4], Maya, Lightwave, 3D Studio Max, etc. They each have a beautiful user interface and have easy drag and drop functionality for the animation design and customization. However, most of them lack underlining programming languages that programmer can program to control various motions. Meanwhile, current motion description languages do not have motion synchronization at the language level. The details of the motion control make the simulation difficult to implement and debug at the language level. This also makes the motions non-reusable.

3. Cybele Language Specification

Our new humanoid motion description language is called Cybele [3]. The features of Cybele language are object-oriented and scripting motion description language. Besides the conventional variable declaration, functions, and control flow, we create a new syntax for describing the parallel actions. We also create a new complex motion statement and expand the use of the conventional motion control concepts using new parallel blocks.

Motion Statements

To describe humanoid motions, we abstracted a collection of predefined motions for the whole body and body parts. We decompose the humanoid in a standard way as specified in humanoid animation specification [5]. Figure 2 shows the BNF of motion statements. In general, a motion statement is composed of a main part, parameter



part, followed by a colon ':', a description part, and ends with a semicolon. A main part includes the object name and the motion name. An object name can be optional. If there is no object name, the system will take the current default object.

A parameter part takes motion parameters that provide additional detail to specify the motion. Each expression is separated by comma ','. The description part can be provided right after the motion function or at the end of a block. It provides a set of attributes to change the scale of the motion. We currently define two attributes namely starttime and speed. We can adapt approaches [12] to retarget and change the scale of motion.

Specification of Complex Motions

Complex motions can be specified as combinations of sequential and/or parallel sub-motions. Sequential motions will execute one after another. Parallel motions will execute in parallel. The start time for parallel motions may not be the same and thus producing partially overlapping motions.

For ease of specification, we defined our language such that programmers can write overlapping sequential and/or parallel motions in compounded program blocks. Each block can be sub-block of other blocks. A compound block can be sequential, parallel, or mixed of the two.

```
[ // begin parallel block
statement1; // statement1 starts from time 0
statement2; // statement2 starts from time 0
statement3 : starttime = t1; // statement3 starts from time t1
] // parallel block end
```

Figure 3. Parallel block

We define the syntax of parallel block as statements enclosed within “[]”. Figure 3 shows an example. All statements inside parallel block are considered being issued concurrently. They all start at the same time unless otherwise specified by the “starttime” parameter. In Figure 3 for example, the syntax “statement3 : starttime = t1” specified that statement 3 starts t1 units of time later than the other two statements.

We define the syntax of sequential block as statements enclosed within “{ }”. Figure 4 shows an example, in which statement2 starts after statement1 completed. A

```
{ // begin sequential block
statement1; // statement1 starts from time t0 (duration=d1)
statement2; // statement2 starts from (t0+d1) (duration=d2)
} // total duration is d1+d2
```

Figure 4. Sequential block

delay can be used such that statement 2 does not need to start right after the completion of statement 1.

```
[
{ statement1; statement2; }
// a sequence begin from time 0
statement3; // statement3 begins at time 0
[ { statement4; statement5; }
// a sequence begin from time 1
statement6; // statement6 begins at time 1
] : starttime = 1
]
```

Figure 5. Complex block

Sequential and parallel blocks can be combined or embedded with each other to describe complex motion logic. Figure 5 shows an example of the structure. The code in Figure 5 illustrates the time specification as

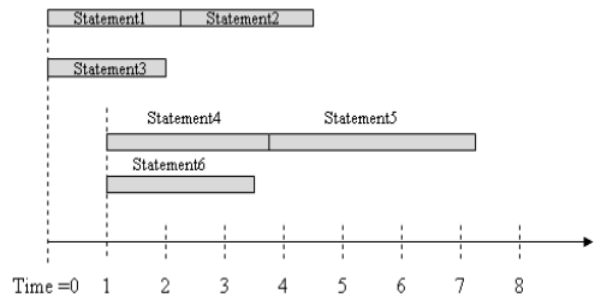


Figure 6. Complex block time slot

shown in Figure 6.

Sample Program

We show a sample Cybele program in Figure 7. In this program, Jack is the actor. He walks backward two steps. Then, he nods his head, while at the same time, he walks forward two steps. Then, he makes a right turn, a left turn, step right once, and step left once in a sequential way.

```
{
Jack.backwalk(2) : speed = 1;
[
Jack.nod(1);
Jack.walk(2) : speed= 1;
]
Jack.turnright();
Jack.turnleft();
Jack.sidewalkright(1);
Jack.sidewalkleft(1);
}
```

Figure 7. Sample program

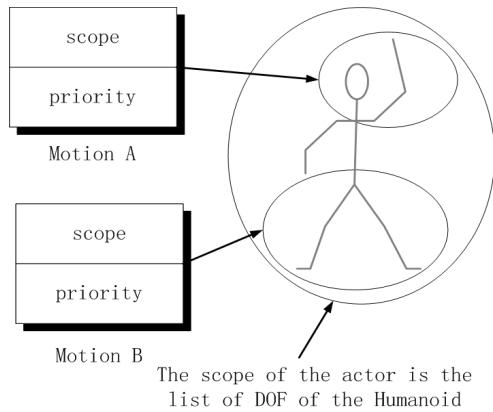


Figure 8. Scope and priority

4. Humanoid Motion Framework in Cybele

To specify humanoid motions, we need to define humanoid framework. Meanwhile, complex motions require checking the constraints and the limitations of humanoid robots. We use H-Anim [8], an international standard, as our humanoid model. We adopt the hierarchy tree structure of humanoids [10], use degree of freedom (DOF) [2] to describe each joint, define states of humanoid, assign priorities to motions, and check constraints and limitations.

Degree of Freedom and Joints

In humanoid robots, joints connect segments through attachment frames that are called sites. Each segment can have several sites. Joints connect sites on different segments within the same figure. Each joint can have at most six degree of freedoms (DOF), including three for rotational and three for translational. In our system, figures have only three rotational DOF, which is defined by the corresponding angle around the axis. Some constraints include rotation axis, joint angle, and upper and lower limits of the joint angle.

States of Humanoid

States are very important aspects of the humanoid model. Static and recognizable states such as stand, sit, crawl, prone, or supine is defined by the relative positioning of each body part, joint and DOF in our system. Programmers can create their customized states by creating new states, which can be a set of DOF for each affected joint. The same state can be applied on different humanoid models, which makes different humanoids move into certain posture.

We describe a partial state of a humanoid as a body part or a set of body parts associated with certain predefined DOF. Partial states are assigned scope and priority

(Figure 8). Scope defines the subset of joints that the motion affected. Priority defines the strength and relative importance of sub-motion with respect to all other motions that are in current or different blocks. The scope and priority allows us to combine partial states to form whole body states or postures, and allows new motions to be generated and synchronized.

State Transition Approach

After we defined states of humanoid, we can define transitions between two states. Transitions between states [1] imply movement from one posture to a new posture. Some transitions are possible and some are not, based on the limitation of humanoid robots. Certain transitions are selected and defined as primitive motions. Some of the motion primitives we defined are walk, run, jump, and turn. Our system takes the motion primitives and automatically generates the complete motion sequence from the initial state to the final state.

Motion Combination with Constraints Checking

When two motions intersect with each other in terms of time and/or scope, we need to check the constraints imposed by the limitations of humanoid robots. For example, we cannot require an arm to move forward at the same time require the arm to move backward. When motions are organized in a sequential way, one after another, and we do not need to check this type of conflicts.

We automated the process of combining parallel motions and resolving possible conflicts between motions. The process uses the predefined scope and priority of each motion. Table 1 shows some examples of priorities that is described here as weights. The weight ranges from 0 to 1 and 1 being the highest. For example, a motion with weight 0 cannot be combined with any other motion within the same scope, while a motion with weight 1 will

Table 1. Combination of Motions Based on Weights

Weight	Descriptions
$\omega = 1$	This weighted motion take highest priority over other motions within the same scope. All other existent motions within the scope are transitioned to finish then this weighted motion starts.
$0 < \omega < 1$	This weighted motion is combined with other weighted motions within the same scope, based on their relative weights.
$\omega = 0$	This weighted motion cannot be combined with any other motion within the same scope. The motion is executed after existing motions finished.

be executed prior to any other motion within its scope. If two motions have the same predefined weight, then the relative position in the program block is used to determine their priorities. In this case, the first statement has higher priority than the second one. To reduce the complexity of assigned detailed weight, we predefined weight categories and grouped motions into the categories.

Three Dimensions of Complex Humanoid Motions

We defined three dimensions for complex humanoid motions. There are time, scope, and level. Figure 9 shows an example of their relationships. The time axis shows the beginning and ending time for each motion. The scope axis indicates which joints or body parts are affected by the motion. The level axis shows the hierarchical structure among various motions nested within complex sequential and parallel blocks. The level is used during motion combination process, in which lowest level motions are created first then process to the highest level.

5. Cybele Humanoid Simulation System

We developed a virtual reality simulator for humanoid

```

{ //Block 1: level 1, attribute 0
... //statements: BlockID 1, level 1, attribute 0
  [ //Block 2: level 2, attribute 1
  ... //statements: BlockID 2, level 2, attribute 1
  ] //Block 2: end
... //statements: BlockID 1, level 1, attribute 0
  [ //Block 3: level 2, attribute 1
  ... //statements: BlockID 3, level 2, attribute 1
    { //Block 4: level 3, attribute 0
    ... //statements: BlockID 4, level 3, attribute 0
      [ //Block 5: level 4, attribute 1
      ... //statements: BlockID 5, level 4, attribute 1
      ] //Block 5: end
    } //Block 4: end
  ... //statements: BlockID 3, level 2, attribute 1
  [ //Block 6: level 3, attribute 1
  ... //statements: BlockID 6, level 3, attribute 1
  ] //Block 6: end
  ... //statements: BlockID 3, level 2, attribute 1
  ] //Block 3: end
... //statement: BlockID 1, level 1, attribute 0
} //Block 1: end
  
```

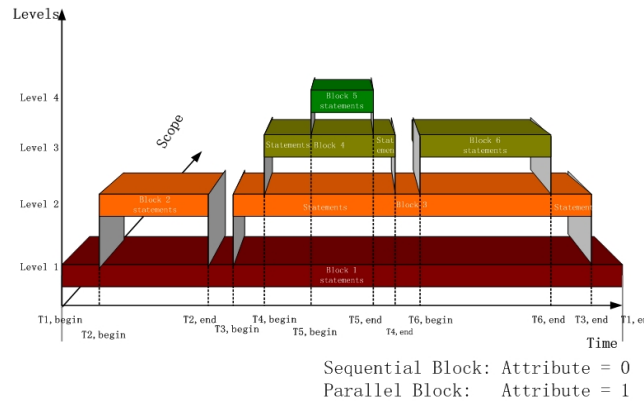


Figure 9. Three Dimensions of Humanoid Motions

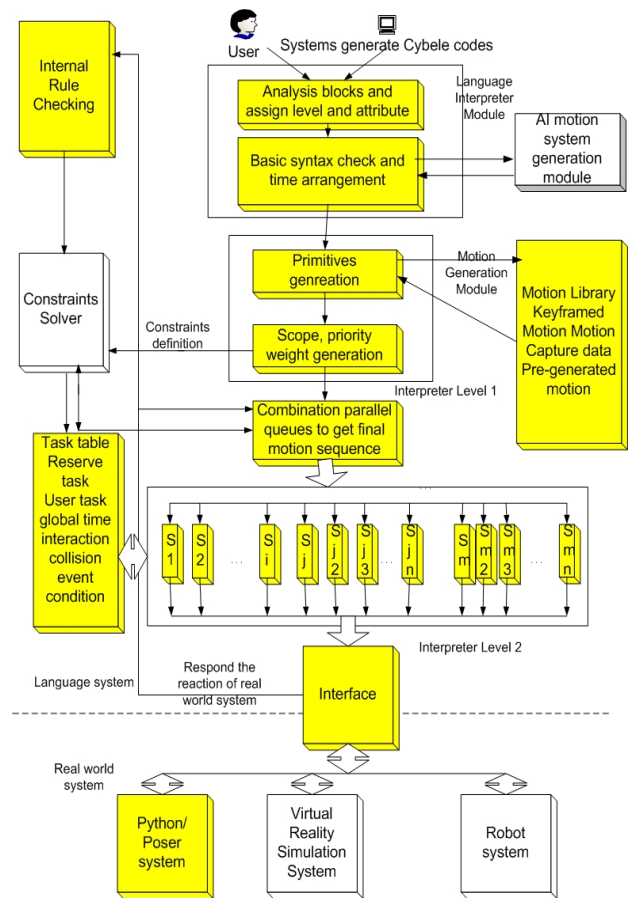


Figure 10. Humanoid Simulation System Overview

robots based on our new motion description language and humanoid framework. The simulation system interprets the motions specified by the language and checks the constraints based on the humanoid framework. The system is designed to interact with user inputs and control humanoid robots in real time.

Figure 10 shows an overview of our humanoid simulation system. The system first gets the Cybele program from some consoles or from other interfacing systems. It interprets the program by analyzing various syntax and structures. Then this language interpreter module passes the results to the motion generation module, which creates motion sequences. In this module, all instructions will be broken down into primitives and tagged with level and block information. Every primitive is generated based on parameters and attributes such as duration, speed, and so on. It performs the following functions:

1. Sub-motion generation. The interpreter checks the library to find the proper primitive sequences.
2. Time schedule. A time scheduler schedules the duration for each sub-motion.

After all the primitives are generated, the motion sequences (labeled S's in the figure) are transferred to the interface. The interface has two functions. The first function is to transmit the elementary motion sequences to various real world systems, which consist of Python/Poser system, Virtual Reality Simulation system, and Robot system (as shown in Figure 10).

The second function of the interface module is to collect the feedbacks and other system parameters from the real world system. These feedbacks are used by the language system to adapt to the current constraints imposed by the environment.

There are two types of constraints in our system. One is an internal rules constraint which is defined by the system and cannot be changed after the environment is created. The other is a rules constraint created by programmers after the environment is created and that can be changed by programmers. All these constraints are resolved by the Constraints Solver module (shown on the right side of the figure).

As can be seen from the system overview (Figure 10), this system is quite complex. Currently, we have completed the implementation and testing of the highlighted modules. Our test results show that such a system is feasible.

6. Conclusion and Future Work

This paper proposes a new motion description language called Cybele. The new features include complex parallel and sequential blocks, new syntax for motion statements, and built-in constraints for motion combination. The motion description links tightly to the underlying models. We defined a humanoid motion framework to make specification of humanoid motion possible. Then we created a simulation system to implement the framework and to execute the motion descriptions. For our system, we also created a solution to generate motion sequences for the complex parallel and sequential blocks with constraints-checking. To solve the motion mixing problem in complex blocks, we present an approach to synthesizing motions with partial states combination. In the solution, scope and priority are defined, and primitives are combined to produce final motions with constraints checking. The result shows that Cybele is an effective motion description language, and also shows that such virtual reality simulation of humanoid robots is feasible.

Our proposed simulation system is quite complex as shown Figure 10, which includes using artificial intelligent techniques to automatically generate detailed motion sequences based on given goal statements. Further work is required to develop such AI motion generation system and to further develop AI methods for solving

various constraints imposed by the limitation of the humanoid robots and by the environments. Although our system is designed to interact directly with humanoid robots [6][15], we have yet to test the system using real robot. Although our virtual simulation shows such system is feasible, a real humanoid robot will encounter much more constraints when interacting directly with real human environments including interacting with human. Thus, future research would also focus on multi-agent systems in which robots, human, and their environments interacting with each others in real time.

REFERENCES

- [1] Badler, N., Bindiganavale, R., Granieri, J., Wei, S., and Zhao, X., "Posture interpolation with collision avoidance," In Proc. Computer Animation '94, pages 13–20, 1994.
- [2] Badler, N., Phillips, C., and Webber, B., "Simulating Humans: Computer Graphics, Animation, and Control," Oxford University Press, 1993.
- [3] Choi, B., and Chen, Y., "Humanoid Motion Description Language," Second International Workshop on Epigenetic Robotics, pp. 21-24, 2002.
- [4] Conway, M.J., "Alice: Easy-to-Learn 3D scripting language for novices," Ph. D thesis, University of Virginia, 1997.
- [5] Harris, J.A., Pittman, A.M., Waller, M.S., and Dark, C.L., "Dance a while Handbook for Folk, Square, Contra and Social Dance," Benjamin Cummings, 1999.
- [6] Hirai, K., Hirose, M., Haikawa, Y., Takenaka, T., "The Development of Honda Humanoid Robot", Proceeding of IEEE Intl. Conference on Robotics and Automation (ICRA), pp. 1321-1326, 1998.
- [7] Huang, Z., Eliëns, A., and Visser, C., "STEP: a Scripting Language for Embodied Agents," Proceedings of the Workshop of Lifelike Animated Agents, Tokyo, 2002
- [8] Humanoid animation work group. <http://www.h-anim.org>.
- [9] Kanayama, Y., and Wu, C., "It's Time to Make Mobile Robots Programmable," Proceedings of Intl. Conference on Robotic and Automation (ICRA), San Francisco, 2000
- [10] Lee, J and Shin, S., "A hierarchical approach to interactive motion editing for human-like figures". In Proceedings of SIGGRAPH 1999, pages 39-48, 1999.
- [11] Perlin, K., and Goldberg, A., "Improv: A System for Scripting Interactive Actors in Virtual Worlds," Computer Graphics; Vol. 29 No. 3., 1996
- [12] Pollard, N. S., "Simple Machines for Scaling Human Motion," Eurographics Workshop on Animation and Simulation, Milan, Italy, 1999
- [13] Python. <http://www.python.org>
- [14] Schrand, R., "Poser 4 Pro Pack f/x & Design," The Coriolis Group, 2001.
- [15] Sony QRIO humanoid robot, http://www.sony.net/SonyInfo/QRIO/top_nf.html, 2007.