INTERNAL REHEARSAL FOR A COGNITIVE ROBOT USING COLLISION DETECTION

By

Joseph F. Hall III

Thesis

Submitted to the Faculty of the

Graduate School of Vanderbilt University

in partial fulfillment of the requirements

for the degree of

MASTER OF SCIENCE

in

Electrical Engineering

December, 2007

Nashville, Tennessee

Approved:

Professor Kazuhiko Kawamura

Professor D. Mitch Wilkes

ACKNOWLEDGEMENTS

First and foremost, I would like to thank Dr. Kazuhiko Kawamura, for his help and guidance throughout this learning experience. Dr. K taught me how to be a better orator, how to be an excellent teaching assistant, and how to research ideas effectively.

Stephen Gordon and Palis Ratanaswasd also deserve thanks. Stephen helped me record new motions for the Verb/Adverb interpolation system for my thesis. Palis helped me integrate my Internal Rehearsal System into his overall cognitive architecture.

I appreciate Dr. Wilkes and Flo Fottrell’s help and support over these last two years. Flo kept me organized and on task. Dr. Wilkes assisted me in forming the foundation of the Internal Rehearsal System in the very beginning of my master’s thesis journey.

I would like to thank my immediate family, specifically my father, mother, and sister for their support. I really could not have done this without their help. With their help and prayers, I was able to finish this thesis.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER I

INTRODUCTION

Objective

The objective of this master's thesis is to explore the concept of robotic internal rehearsal, and to determine how this concept can be used to assist ISAC in performing cognitive tasks. If successful, this will allow the robot to contemplate the consequences of its actions before they are performed by the actuation system. In other words, the robot predicts its future state and the future state of the surrounding environment by emulating the action internally. The robot's cognitive architecture takes this prediction and uses it to influence the system's actions.

Important Concepts

There are two main methods for action prediction used within robotics. The most popular method for robotic action prediction involves forward modeling. The forward model approach allows robots to directly couple motor commands and predicted environmental observations given the robot's current state (figure 1) [Dearden et. al, 2005]. Forward models can be used individually as stand-alone predictors, or in more complex situations, they can be hierarchically organized or arranged in clusters to gain predictive knowledge about the system itself [Wolpert and Kawato, 1998].

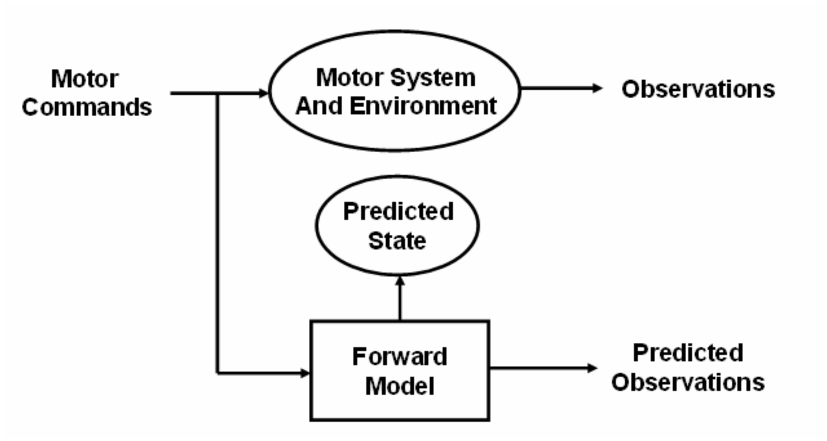


Figure 1: Robotic Forward Modeling [Dearden et. al, 2005]

Forward models are trained networks such as Bayesian networks, neural networks, or radial basis function networks that learn how to correlate a given motor command to a predicted action or observation given the current state [Dearden et al, 2005; G. Sun and B. Scassellati, 2004]. When a forward model is fully trained, it can be inverted into an inverse model. This model performs the opposite service; it outputs the motor command given the current robotic state and desired observation. The inverse model is used to control the system from that time forward.

The second method for action prediction is called robotic internal simulation. This approach uses an "inner world" that allows the robot to internally perform the action and determine results from the consequences. This research is based on human internal simulation based on "simulation hypothesis" [Hesslow, 2002]. Hesslow's work analyzed a human's ability to internally simulate behavior, internally simulate perception, and to anticipate what will happen next. This concept will be further explored in this thesis. There are some individuals working to integrate internal simulation into cognitive robotics, namely Murray Shanahan of the Imperial College in London. His research involves the use of a mobile robot based on Global Workspace cognitive architecture with two sensorimotor loops: the reactive response loop or 'first-order' loop, and the cognitive response loop or ‘higher-order’ loop [Shanahan, 2005]. Internal simulation plays a critical role in allowing the cognitive higher-order loop to suppress the reactive loop when the reactive loop is about choose a poor course of action. This concept is further explored in the Cognitive Architectures section of Chapter II. This master’s thesis focuses on creating a robotic internal simulation like system for ISAC the cognitive robot. This system is called the Internal Rehearsal System or IRS. Forward modeling related research may be pursued some time in the future.

Organization of Thesis

The thesis is organized in the following manner: Chapter II contains background information about cognitive robotics and cognitive robotic architectures. It describes a robotic arm collision avoidance developed for ISAC at Vanderbilt, and finally, this section discusses the robot ISAC and its multi-agent cognitive architecture. Chapter III discusses how the Internal Rehearsal System is designed and how it fits into ISAC cognitive architecture. Chapter IV discusses the experiments and their significance, while Chapter V discusses the results from those experiments. Chapter VI contains the conclusion along with future work from this thesis.

CHAPTER II

BACKGROUND INFORMATION

Cognitive Robotics

The concept of Cognitive Robotics comes from cognition in humans. Humans have the ability to exhibit cognitive control from a part of the brain called the pre-frontal cortex. This part of the brain allows humans to modify or suppress reactive behaviors to achieve future goals that align with the human's intentions. These goals are often located far into the future and require behavior that is planned and well orchestrated [Miller, 2000].

Cognitive control's most useful aspect has to do with learning from past experience. The pre-frontal cortex allows humans to extract key features of a task, and the neural tissues within the prefrontal cortex allow these features to stored and recalled at a later date. For instance, humans are not born with table manners. They must learn how and when to perform certain behaviors, such as not eating until everyone is served even though one might be hungry. According to Miller, cognitive control is an important functionality for intelligent behavior generation [Miller, 2000].

Robotic cognitive control is a reflection of its human counterpart. Robotic cognition is heavily dependant on the concept of cognitive or autonomous agent. "An autonomous agent is a system situated within and a part of an environment that senses that environment and acts on it, over time, in pursuit of its own agenda and so as to effect what it senses in the future" [Franklin and Graesser, 1997]. Franklin, however, uses a rather broad definition for cognitive agent. Cognitive agents can be many things like biological creatures, robots, or intelligent software-based entities [Franklin, 1997]. This thesis is, of course, interested in cognitive robots.

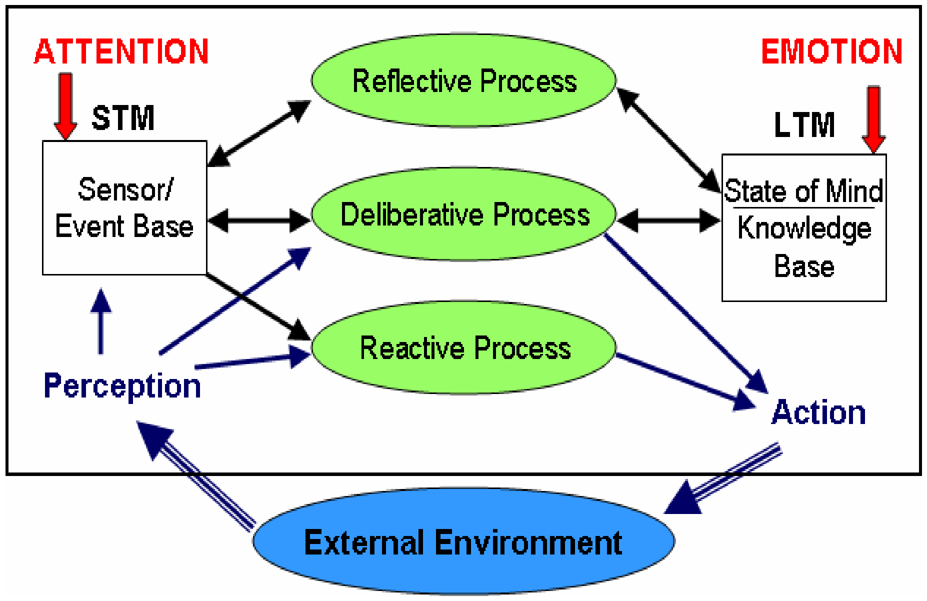


Figure 2: Cognitive Robotic System Concept [DARPA, 2002 (modified by K. Kawamura)]

Figure 2 shows some important concepts in cognitive robotic systems. Cognitive robots have the ability to sense the environment, ability to change the environment, possess memory structures (like short-term and long-term memory), and different processes that take sensory input and produce actuator output. These processes are the most important to proper cognitive control.

The bottom process is known as reactive control in figure 2. Reactive responses are executed automatically. They require very little thought to occur, and they happen very rapidly. Deliberative responses are different and take a longer period of time. These responses involve planning and reasoning with past knowledge to solve the task at hand. In most cases, deliberative responses are dependant on internal models as well [Franklin, 1997]. The top process is the slowest of all. Reflective processes involve the robot thinking about itself. In other words, how does the robot improve itself, its behavior, and its reasoning ability? Self reflective processes can also involve self diagnosis and solving problems by using completely novel approaches [Kawamura, 2003].

One important concept in cognitive robotics is embodiment [Dodd, 2005]. Human brains are not general use computers; they have an accompanied body to control. Cognitive robots should also incorporate this fact as well for one main reason; Cognitive robots interact with the environment. In order to do this, the robot must be able to perceive objects and perform actions on the surrounding environment. One example of cognitive robotic embodiment is shown in figure 3.

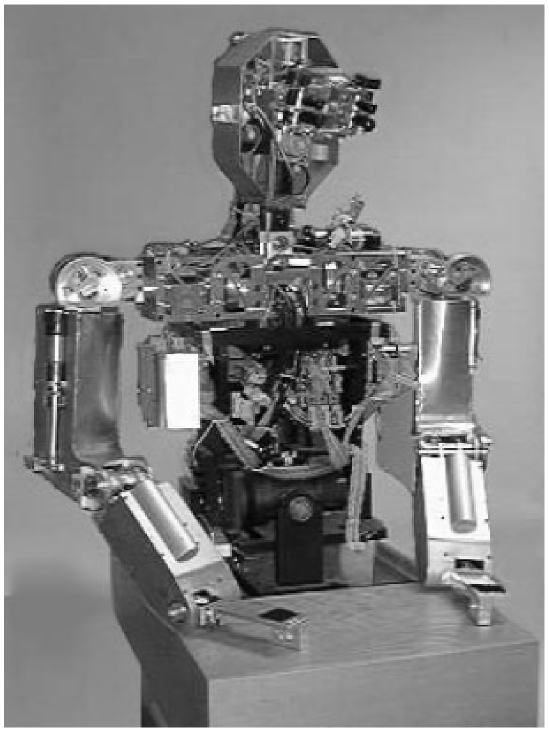


Figure 3: MIT’s Cog the Cognitive Robot [Brooks, 1998]

This figure shows MIT’s cognitive robot called “COG”. This robot’s cognition was built on four main principles [Brooks, 1998]:

1. Gradual improvements in cognition
2. Interaction with humans
3. Embodied intelligence
4. Synthesizing sensory information

Cog uses the concept of embodied intelligence to make computation easier. For instance, [Brooks, 2007] shows Cog sawing a board. The exercise appears to use very complex inverse kinematics, but the arm actuators are simply oscillating while sawing. This embodied intelligence allows Cog to produce a complex behavior by performing a simple motion. Vanderbilt's cognitive robot, ISAC, does not currently use embodiment to this extent. Instead, ISAC uses embodiment to simply perceive from and act on the environment.

The next section shows four cognitive architectures. Some of them are very well known and well established like ACT-R and SOAR. These two are similar because they use symbolic production systems to analyze and generate human-like cognitive behavior [Budiu, 2007; Laird et. al., 1987]. The two others architectures are in various stages of development. These two architectures, IDA and Shanahan's GW Cognitive Architecture, are both applications of Global Workspace Theory [Franklin et al., 1998; Shanahan, 2005; Baars, 1997].

Cognitive Architectures

ACT-R Cognitive Architecture

ACT-R is an acronym for “Adaptive Character of Thought –Rational”, and it is a cognitive architecture which measures the performance of human subjects during cognitive exercises [Anderson, 1996]. This architecture is built upon human psychological responses during previous cognitive experiments. Because of this, the architecture can learn how to emulate a human’s response during a cognitive task. Thus, the performance of the system can be directly compared to human performance [Budiu, 2007].

ACT-R has three main types of subsystems: modules, buffers, and pattern matchers. Module subsystems can be of two types: perceptual-motor modules and memory modules. Perceptual-motor modules are responsible for interfacing the cognitive system with the outside environment. These modules can be seen in figure 4 as the “Visual Module” and the “Motor Module” which encode percepts and generate actions respectively [Budiu, 2007].

Memory modules can be of two types: declarative memory and procedural memory. Declarative memory is long-term memory which contains facts learned by the system such as a stop sign is red, grass is green, etc… The declarative memory stores these facts as data structures known as chunks. Procedural memory is also a long-term memory, and it deals with learning how to perform an action, like how to ride a bicycle or tie a shoe. These pieces of procedural memory information are stored in data structures called productions. They are essentially IF-THEN rules like IF a percept is present THEN perform the appropriate action [Budiu, 2007].

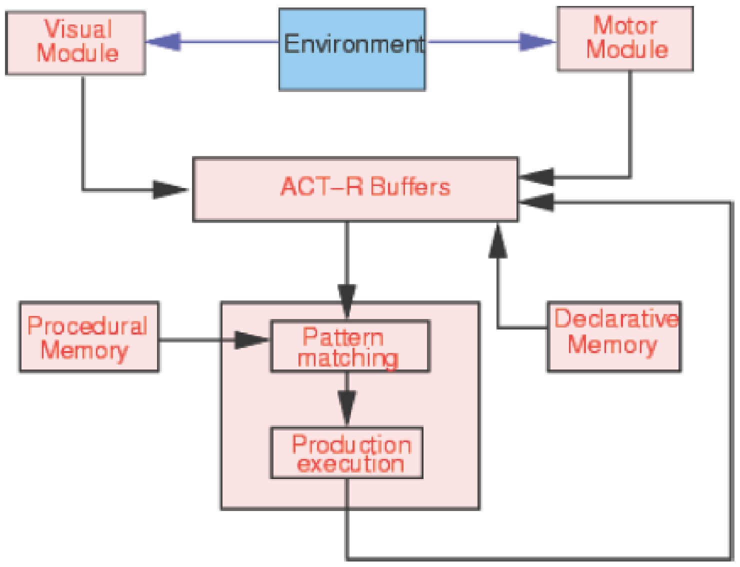


Figure 4: ACT-R Cognitive Architecture [Budiu, 2007]

Buffers are the communication lines for the system. ACT-R is able to communicate with each module via buffers, with the exception of procedural memory. Buffers are also useful in that the complete state of the system can be determined at any given time from the contents of the buffers [Budiu, 2007]. Also, according to [Anderson et al., 2004], the buffer system can be considered the working memory system of this architecture.

Finally, the pattern matcher is the workhorse of this system. The pattern matcher will check the state of the buffers at each time step and determine which production rule is most appropriate. If a production rule matches the buffer state, that production rule will fire. This firing may change the contents of the buffers by changing the external environment through the motor module or the goal of the system. However, only one production rule can fire at a time, so the system can be traced easily. So, basically, this architecture's cognition is produced by a series of production firings [Budiu, 2007].

SOAR Cognitive Architecture

SOAR is a goal-based cognitive architecture which strives to create complete machine intelligence. First, SOAR can work on a spectrum of problems from trivial situations to open-ended and difficult conundrums. Secondly, SOAR can use a myriad of problem solving tools to accomplish the task at hand. And lastly, SOAR can gauge its performance during the task [Laird et al., 1987]. A high-level system diagram of the entire SOAR Architecture can be seen in figure 5 below.

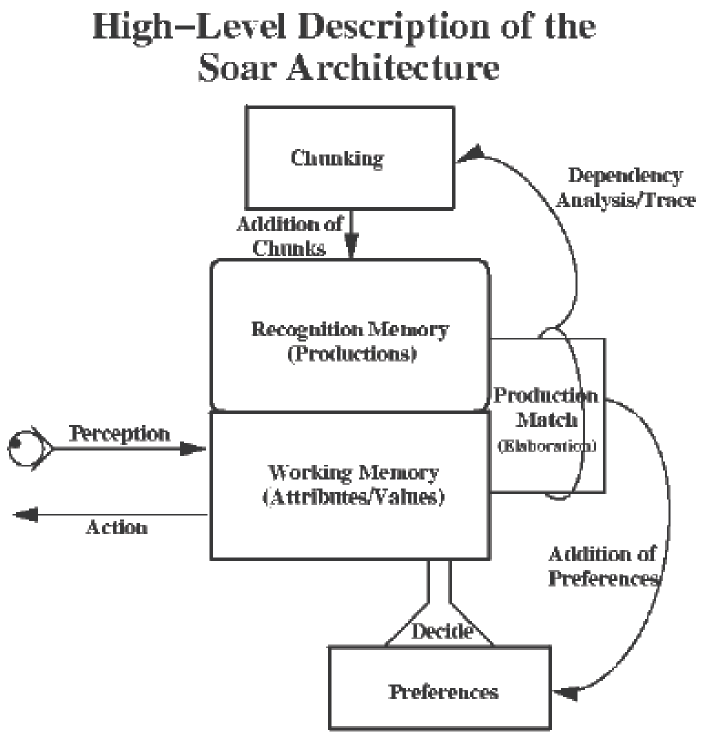

High–Level Description of the Soar Architecture
Chunking
Dependency Analysis/Trace
Addition of Chunks
Recognition Memory (Productions)
Production Match (Elaboration)
Perception
Working Memory (Attributes/Values)
Action
Addition of Preferences
Decide
Preferences

Figure 5: SOAR Cognitive Architecture [modified from Wray et al., 2005]

SOAR, like ACT-R, has a long-term memory based on production rules; the production rules are of the IF <consequent> THEN <antecedent> rule form. The working memory system, on the other hand holds important task knowledge in the form of attribute/value data structures. These productions, which match the attribute/value chunks in working memory, fire in parallel, and in the process adds to the contents of the working memory system. This will cause other productions to match and fire; this recursive process will continue until no other production rules match. This process is known as the elaboration phase of execution [Laird et al., 1987].

When SOAR has difficulty solving a problem, the architecture will create goals to correct itself. This is called "impasse resolution". SOAR will create a problem space and a set of operators to find a solution within the space to achieve the desired goal. Subgoals can also be created during this process to solve the problem. When the impasse is resolved, the methods used to solve the impasse are transformed into production rules. This transformation is known as chunking. This is how SOAR implements learning [Laird et al., 1987].

The last part of SOAR's architecture is the preference system. The preference system sorts the firings from the production rules and determines a preferred action based on which action will produce the best result. This preference system eliminates the poorly chosen actions and chooses the best action for the given situation. This process is known as the decision phase of execution [Laird et al., 1987].

IDA Cognitive Architecture

IDA is a cognitive multi-agent architecture specifically tailored for the U.S. Navy for personnel assignment. IDA is being developed at the University of Memphis to replace a specific naval officer called a detailer. These detailers are responsible for assigning navy personnel based on the needs of the Navy at any given time. IDA, when complete, will take over the assigning process.

IDA is different from both ACT-R and SOAR in that the architecture is based on Global Workspace theory developed by Baars [Baars, 1997]. The Global Workspace allows specialized subconscious processes to compete against one another to become the conscious process. This competition can be seen in figure 6.

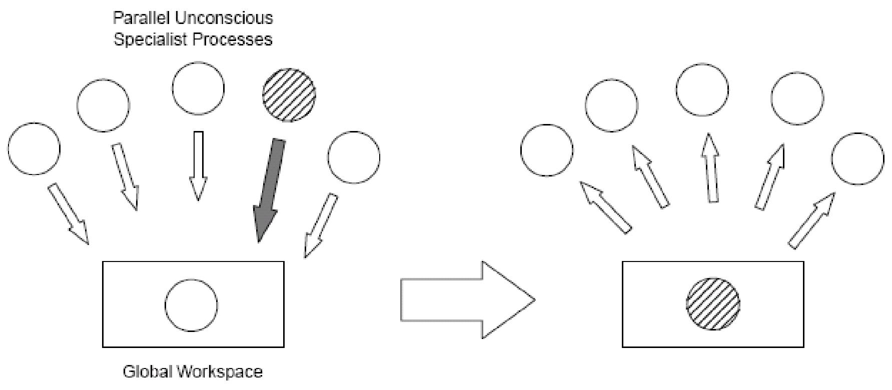


Figure 6: Global Workspace Theory in Action [Shanahan, 2005]

Once a subconscious process wins the competition, this process broadcasts its conscious status to the other processes. This broadcast informs the other processes to help the conscious process with the current task. When this process is done, the consciousness competition will begin anew, and another process will become conscious [Baars, 1997].

IDA receives input from two sources, email messages and information from select databases. IDA perceives information from these sources by utilizing a knowledge base coupled with a workspace. The knowledge base is represented by a specialized neural network called a slipnet, and the workspace performs a human-like working memory service for the slipnet. This perception architecture is based on perceptual Copy Cat Architecture [Hofstadter and Mitchell, 1994]. IDA also requires an action selection system. IDA performs actions based on the output of IDA's Behavior Net system [Maes, 1989]. Lastly, the selection knowledge base along with the focus module handles the cognitive or deliberative aspect of the architecture. The selection knowledge base gives IDA the knowledge of how to select which sailors for which job. The focus model is the method in which IDA understands herself and the outside world [Franklin et al., 1998].

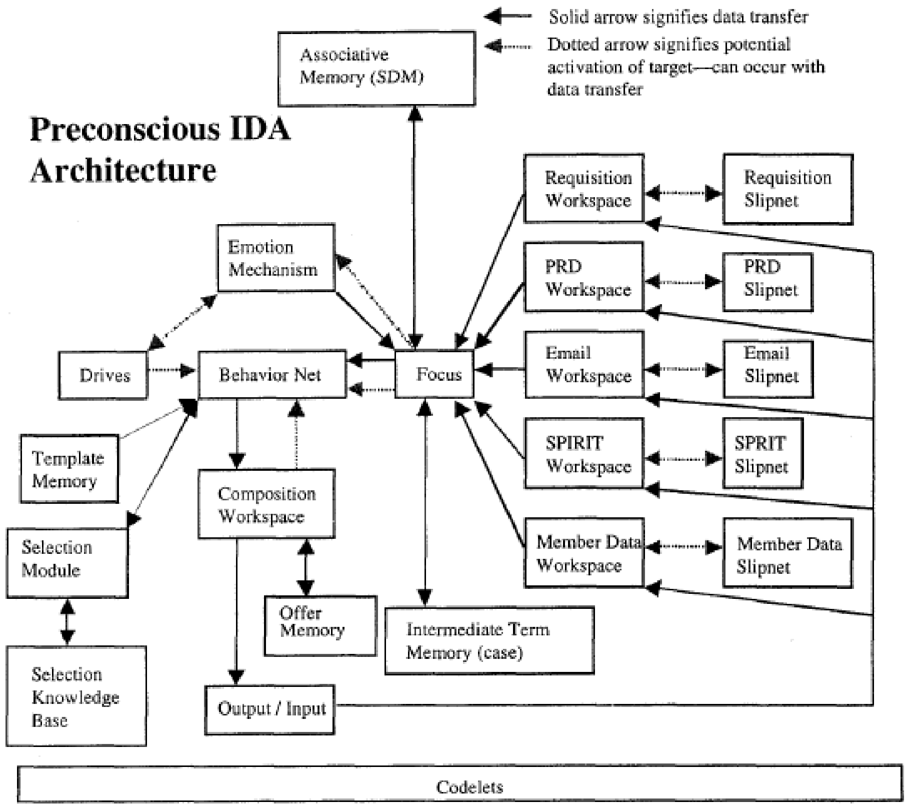


Figure 7: IDA Cognitive Architecture [Franklin et al., 1998]

The entire high-level system described above is built by using low-level codelets, which are small specialized processes that work together according the Global Workspace theory approach. This system can be seen in figure 7.

Shanahan's Cognitive Architecture

Murray Shanahan utilizes Global Workspace theory [Baars, 1997] and internal simulation to create a cognitive architecture which mirrors the human brain. This architecture models the sensory and motor cortices, the basal ganglia (for action selection), the amygdala (for decision making), and the thalmacortical and cortocortical areas of the brain. This architecture gives rise to three important human psychological properties: consciousness, imagination, and emotion [Shanahan, 2005].

Global Workspace theory produces consciousness by allowing attention-getting competition between subconscious processes. This has been previously discussed and is illustrated in Figure 6 [Shanahan, 2005].

Imagination is also important to this architecture. Imagination allows the intelligent system to try an action internally before it is executed in real life. This allows the system to avoid bad mistakes by internally evaluating the consequences. This concept is of central importance to this thesis [Shanahan, 2005].

Finally, emotion performs mediation on the system's decisions. This architecture uses emotion for mediating decision making and action selection. Emotion allows the system to determine if one course of action is better than another [Shanahan, 2005].

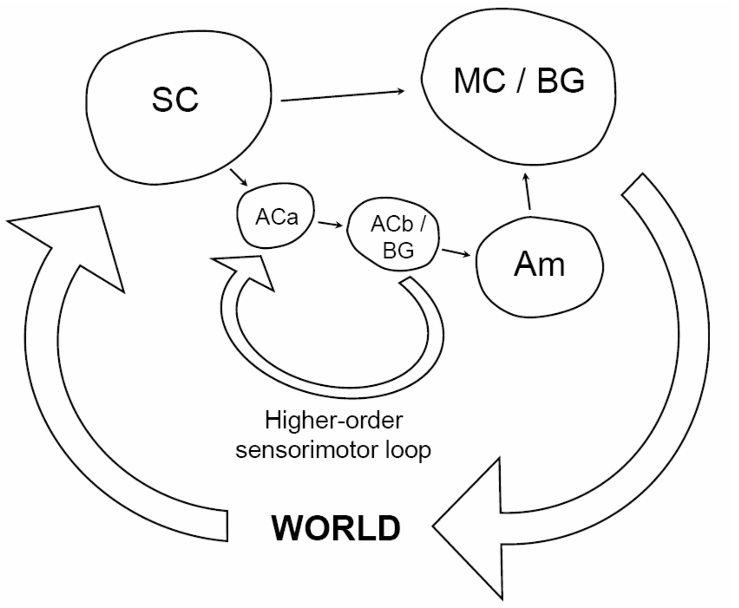


Figure 8: "Top-Level" Schematic of Shanahan's Cognitive Architecture [Shanahan, 2005]

Figure 8 shows the “top-level” schematic of the cognitive architecture proposed by Shanahan. There are two sensorimotor loops involved. The outer loop is the “first-order” loop which is the reactive or routine part of the architecture. This loop consists of two parts: the sensory cortex (SC) and the motor cortex/basal ganglia (MC/BG). SC is responsible for recognition of percepts, while MC is responsible for activating appropriate actuators given the decision from the basal ganglia. This “first-order” loop maps sensory data to motor commands using if-then rules. The basal ganglia holds this if-then response in veto for the amygdala’s decision input [Shanahan, 2005].

The inner loop in figure 8 is the “higher-order” loop. This loop is where imagination occurs. This inner loop plays and replays actions and determines the consequences for each. After the results are accumulated, the amygdala looks at the consequences and communicates with the basal ganglia. The basal ganglia then determines the best action from the amygdala’s recommendations [Shanahan, 2005].

The “higher-order” loop has two parts: ACa and ACb. These two modules correspond with the prefrontal cortex of the brain (working memory, planning, cognitive control), and they mirror SC and MC respectively. The difference is that ACa and ACb are involved in imagination whereas SC and MC are involved in actual action selection and execution [Shanahan, 2005].

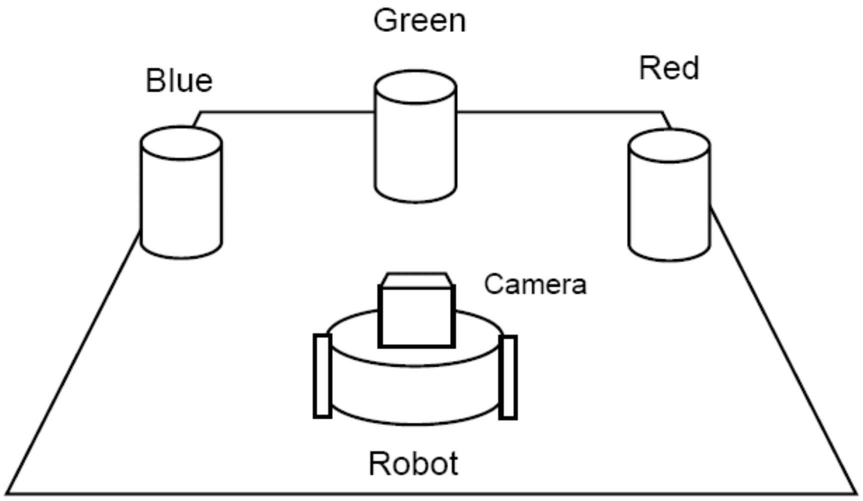


Figure 9: Global Workspace Example Featuring a Mobile Robot [Shanahan, 2005]

An example of this architecture is demonstrated in figure 9. This scenario shows a mobile robot with a blue cylinder to the left, a green cylinder ahead, and a red cylinder to the right. The robot can choose any of the three cylinders by turning or rolling towards it. When the robot is activated, SC, by means of an on-board camera, sees the green cylinder and passes this information to MC/BG. MC looks at an internal lookup table for seeing a green cylinder. The most appropriate reactive response for this sensory data is to rotate right. This response is given to BG and held in veto until the higher-order loop is finished. The higher-order loop then simulates "rotate right". This loop realizes that rotate right will make the robot view the red cylinder (which is perceived as negative by the robot). Therefore, this action will be crossed off from the list. Rotate left is then tried. This yields the robot viewing the blue cylinder. This cylinder is good, so the amygdala tells the basal ganglia to forsake rotate right and use rotate left. The robot will rotate towards the blue cylinder as a final result [Shanahan, 2005].

Three-Dimensional Collision Detection

This section discusses previous research done by Charoenseang [Charoenseang, 1999]. In this research he created a simulator involving collision detection and force feedback for ISAC. The purpose of this work was to allow ISAC to perform tasks utilizing two arms. When ISAC works on a given number of tasks, this system will protect ISAC's arms by not allowing them to run into obstacles or into each other. The simulator is shown in figure 10 [Charoenseang, 1999].

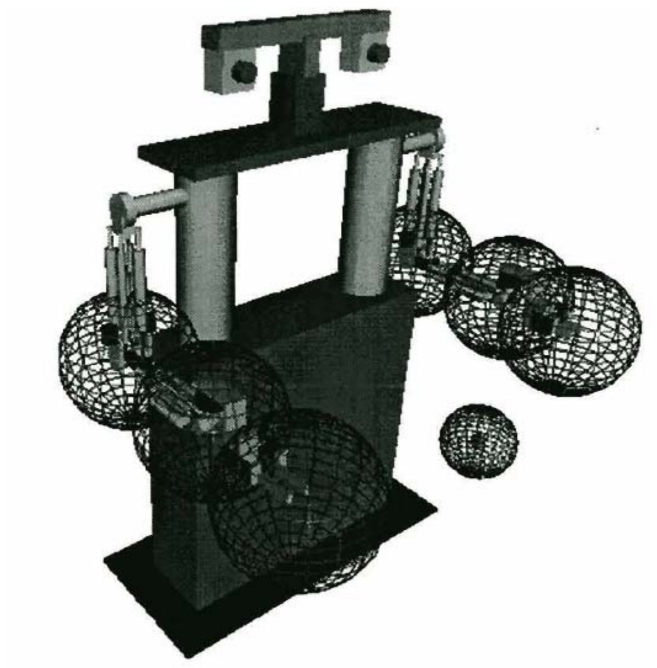


Figure 10: ISAC Collision Detection Simulator with Force Feedback [Charoenseang, 1999]

The simulator has spheres around the manipulator, wrist, and elbow of ISAC's arms, and any obstacle that the robot perceives also has a sphere protecting it. The purpose of these spheres is to create virtual impedance to motion when a collision occurs. According to Charoenseang, a collision occurs when two spheres touch. This causes the distance between the two spheres to be less than the sum of the radii of the spheres. When a collision happens, a value known as the obstacle point X_{obs} is calculated. This point is defined as the closest part of the incoming colliding sphere that is located on the line connecting the centers of both spheres involved in the collision. A diagram of a collision involving virtual impedance can be seen in figure 11. This figure shows a simulated version of ISAC's arm colliding with an obstacle point located at X_{obs}.

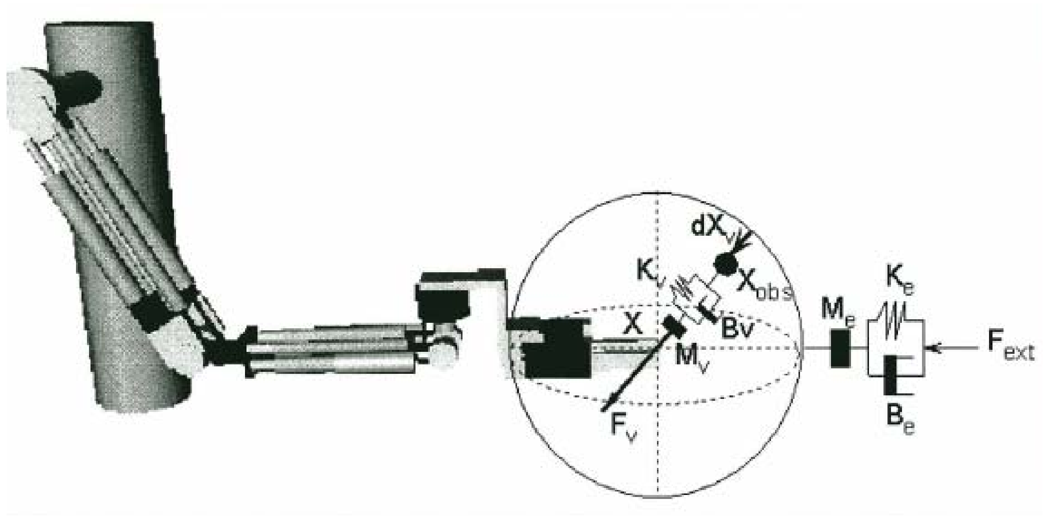


Figure 11: Example of Virtual Impedance Control [Charoenseang, 1999]

When the collision with the obstacle occurs, virtual impedance is initiated that drives the arm away from the obstacle. The virtual impedance is governed by three functions as shown below [Charoenseang, 1999]:

$$M_e d\ddot{X}_e + B_e d\dot{X}_e + K_e dX_e = F_{ext} + F_v \qquad (1)$$

In equation 1, M_e, B_e, and K_e are the mass, dashpot, and spring coefficient matrices for the end effector sphere. Also, in figure 11 and equation 1, dX_e is the distance from the desired position to the end effector. F_{ext} is the external force put on the system; most of the time, the external force will be zero. F_v, or virtual force, is the final value in equation 1. It is calculated via equation 2.

$$F_v = \begin{cases} M_v d\ddot{X}_v + B_v d\dot{X}_v + K_v dX_v & \text{if } X_{obs} < R \\ 0 & \text{otherwise} \end{cases} \quad (2)$$

$$dX_v = \begin{cases} X_v - R \cdot \overline{X}_v & \text{if } X_{obs} < R \\ 0 & \text{otherwise} \end{cases} \quad (3)$$

F_v is the force caused by the difference between the obstacle position and the wall of the end effector sphere. This distance is also known as dX_v, and it is clearly shown in figure 11 and defined by equation 3. $\overline{X}_v$ in equation 3 is a normalized unit vector of X_v. This distance, dX_v, only exists if the position of the obstacle point is within the sphere of the end effector, otherwise it is zero. As formula 2 shows, a force is caused when an obstacle enters this sphere. It is defined by the dX_v and the predefined mass, dashpot, and spring constants Mv, B_v, and Kv respectively [Charoenseang, 1999].

ISAC, A Cognitive Robot

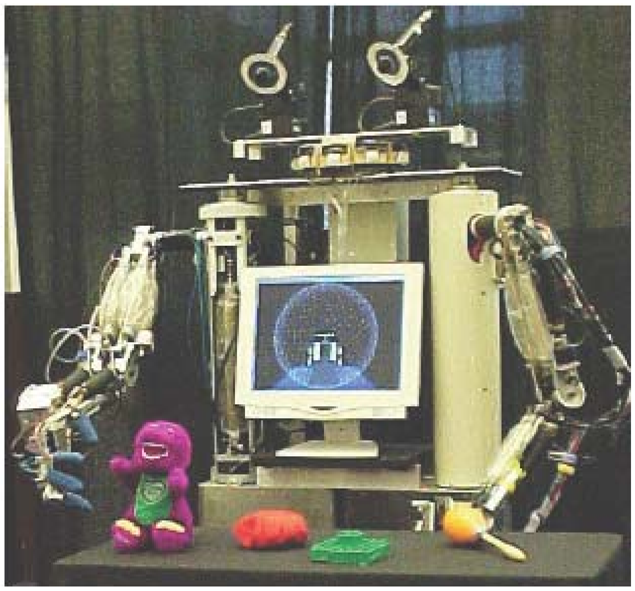


Figure 12: Intelligent Soft Arm Control Testbed [Dodd, 2005]

ISAC stands for Intelligent SoftArm Control, and was designed by the Center for Intelligent Systems at Vanderbilt University. This humanoid robot, pictured in figure 12, was originally designed to assist feeding physically challenged individuals [Kawamura et al., 1995]. ISAC has grown from this original purpose into a testbed for cognitive control-related experiments. ISAC has two cameras perched atop two servo motors for stereotopic vision, along with a laser scanner for determining direction of motion. It has two arms erected from mckibben actuator muscles on an aluminum skeleton [Bridgestone Corp., 1986], and lastly, a hearing and speech system which can understand a limited set of human commands and speak when necessary.

The cameras can move independently of each other; however, in most cases, the cameras work together to determine where an object is by using a stereotopic vision system similar to humans. Currently, ISAC can detect objects by utilizing a color histogram method, and it is able to recognize faces via OpenCV [Rojas, 2004].

The robotic arms consist of twelve Mckibben actuator muscles attached to an aluminum skeleton. The arms are mirror images of each other and are very similar in their forward and reverse kinematics. The Mckibben muscles are arranged in pairs where one is flexed and the other is stretched [Bridgestone Corp., 1986]. This allows ISAC to move in a manner similar to a human, and it also allows the robot's arms to be flexible. The right arm has a hand which allows ISAC tactile sensing capability via proximity sensors [Christopher, 1998]. The left arm is being outfitted with a gripper. This will allow ISAC to grasp objects when necessary.

IMA Multi-Agent Architecture Overview

ISAC's computational environment is a multi-agent architecture called IMA or Intelligent Machine Architecture [Olivares, 2003; Olivares, 2004; Pack, 1998]. This architecture consists of groups of "atomic agents" which work together to form a cognitive system. The concept of the atomic agent came from Minsky's *Society of Mind* [Minsky, 1985]. In this book, Minsky calls brain processes "mental agents" where each process is unintelligent in itself, but the compilation of these processes working together creates intelligence. These atomic agents are different from Franklin's concept of cognitive agent [Franklin and Graesser, 1997]. A cognitive agent is a complete intelligent system; on the other hand atomic agents are pieces of the complete cognitive system.

IMA contains a variety of different atomic agents. Some agents are involved in perception encoding and motor control where each atomic agent represents a hardware or computational resource. However, there are some agents which are actually a compilation of atomic agents. These type of agents are called "compound agents". ISAC currently one agent like this called the Self Agent (shown in figure 13). The Self Agent is very important for performing cognitive tasks and will be discussed later in this section.

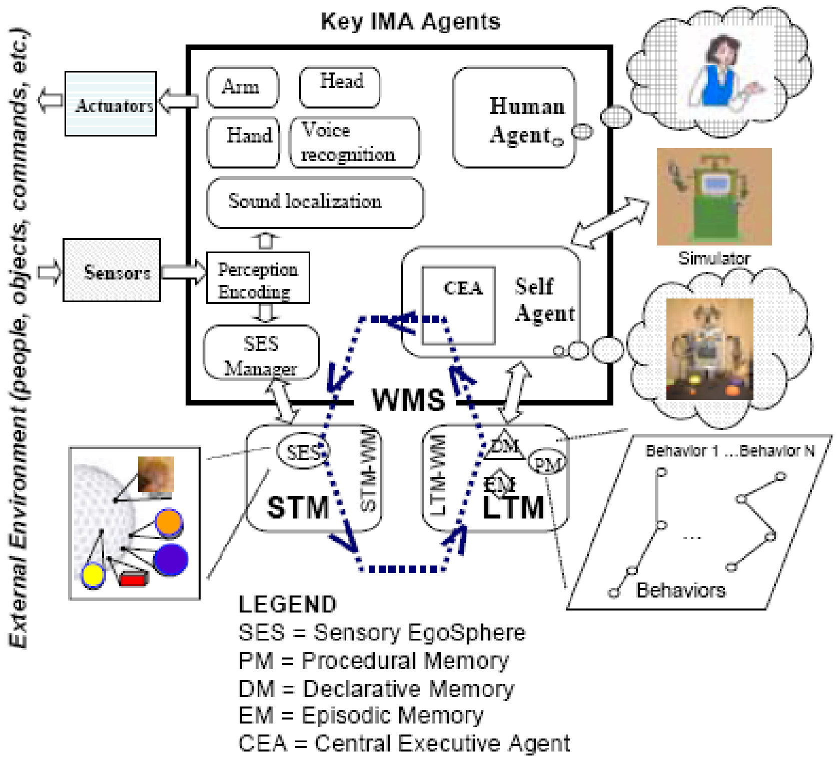


Figure 13: Multi-Agent IMA Architecture for a Cognitive Robotic System [Kawamura et al., 2005]

IMA actually runs on a group of computers controlling various parts of ISAC. These computers run on Windows 2000 or XP and communicate via Microsoft's COM/DCOM technology. IMA's true advantage comes in with code reusability and reconfiguration. In order to create agents, one creates components via Visual C++ and links these components together to form an atomic agent in a GUI program called DAD or Distributed Agent Designer [Olivares, 2003]. These agents can then be formed together to create compound agents and in the complete scope, ISAC's complete cognitive system.

I/O Atomic Agents

ISAC has atomic agents which process input from the perception system and produce output to the motor actuation system. These agents represent hardware sensors and actuators that ISAC can use to interact with the outside environment.

The perceptual system involves multiple atomic agents running in parallel [Rojas, 2004]. The vision agents use ISAC's two stereotopic head cameras as discussed previously. These agents can detect objects via color histograms along with face and motion using OpenCV C++ libraries. ISAC has another motion detection agent which uses a laser scanner to determine where motion is occurring in the environment. And lastly, the sound-based agents can both recognize voice and determine its direction. These agents, while running in parallel, post their information to ISAC's short-term memory system otherwise known as SES or the Sensory Egosphere [Achim, 2005].

ISAC's motor control system involves agents that control the robot's head, arms and hands. The head controls two pan-tilt units that contain cameras for vision. The head agent accepts desired pan-tilt angles for both cameras and changes the servo motors to meet these goal angles. This agent is crucial for visual tasks such as tracking objects and looking for faces and motion. The right and left hand agents allow ISAC to perform grasping functions. ISAC can grasp with its Pneuhands when an object is detected by the palm proximity sensors or when an object is detected by the finger-tip touch detectors [Christopher, 1998]. Finally, ISAC's Arm Agents are responsible for moving ISAC's rubbertuator-based arms to the correct position in either joint or Cartesian space. These agents, right and left arm, contain a non-linear PID controller that moves the arm into position by reducing the joint angle error between the desired angles and actual angles. The output of this controller is a change of pressure in the Mckibben actuator muscles. The muscles are arranged in pairs. In these pairs, one muscle will contract while the other stretches. The change in pressure will cause a muscle to either contract (high pressure), or stretch (low pressure). Because the muscles are in pairs, one must contract while the other stretches. This process allows ISAC to change its joint angels. It is further discussed in [Schroder, 2003].

Self Agent

ISAC's Self Agent is the compound agent that represents ISAC's sense of self. Functions such as cognitive control, task planning, and self monitoring are all processed in this compound agent [Kawamura et al., 2007a]. Cognitive control is the most important function carried out by the Self Agent; "The human brain is known to process a variety of stimuli in parallel, ignore non-critical stimuli to execute the task in hand, and learn new tasks with minimum assistance. This process is known as executive or cognitive control" [Kawamura et al., 2006]. These three functions are carried out by its atomic agents as seen in figure 14. The Self Agent receives task commands interpreted by the Intention Agent. From this command, the Self Agent uses emotion, past experience, internal state variables to make decisions. As with Shanahan's architecture, the Self Agent contains two main loops. A reactive loop is represented by the First-Order Response Agent, and a deliberative or cognitive process is represented by the Central Executive Agent (CEA). The First-Order Response Agent (FRA) contains percept-behavior pairs which automatically fire when a percept is present. FRA is the reactive component of the Self Agent. CEA however, contains the cognitive aspects of the system which involve planning, reasoning and internal rehearsal [Kawamura et al., 2007a].

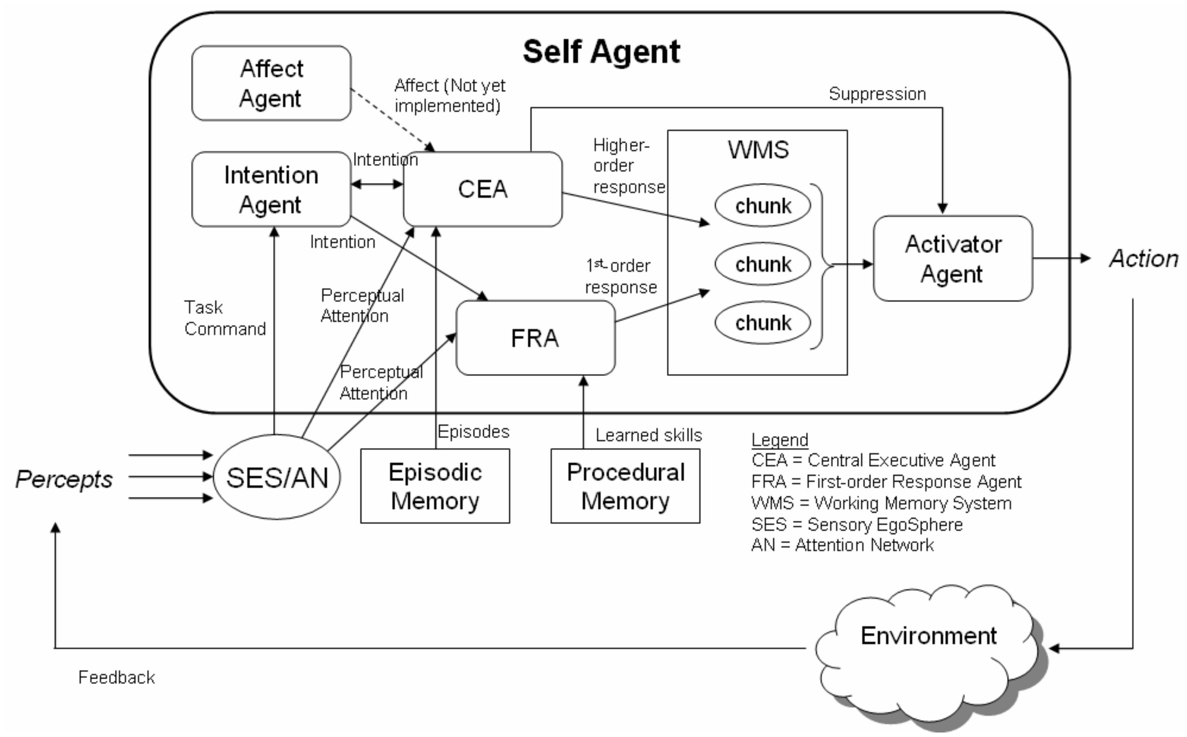


Figure 14: ISAC Self Agent Architecture [Kawamura et al., 2007c]

Activator Agent

Figure 14 shows a high level schematic of the Self Agent. It consists of five atomic agents. The first agent is the Activator Agent (AA). The purpose of AA is very straightforward. It allows the Self Agent to send actuator commands to the various motor control atomic agents. AA will send a command to a motor control atomic agent, and after a set period of time, the atomic agent will report success or failure back to the AA [Kawamura et al., 2007a].

Affect Agent

The purpose of the Affect Agent is supplement cognition via emotion-based state variables. This agent is based on Haikonen's SRTE or Systems Reactions Theory of Emotions [Haikonen, 2003]. The Affect Agent will utilize simple emotions (otherwise known as "elementary sensations") such as pain, pleasure, good, and bad [Dodd and Gutierrez, 2005].

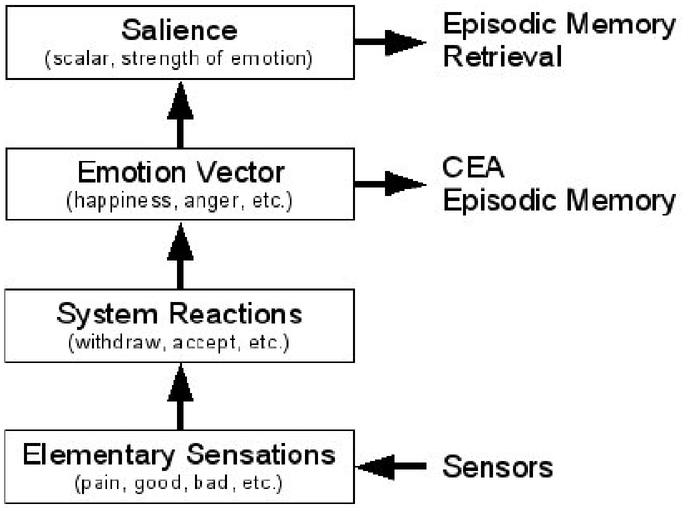


Figure 15: Affect Agent Concept [Dodd and Gutierrez, 2005]

For instance, pain could be triggered by ISAC's joint angles being at or outside of their limits; pleasure could be triggered by adequate lighting and when ISAC receives reward for a job well done. Good and bad could be triggered if either the human supervisor tells ISAC to associate good or bad with an object, or if the interaction with a human or object went well or not [Dodd and Gutierrez, 2005].

The Affect Agent combines the lowest sensations into the next level of "system reactions". For instance, pleasure and bad combined would force ISAC to withdraw. These reactions would be combined to create actual emotions like fear, happiness etc. along with a salience or strength of emotion value. This concept is shown in figure 15 [Dodd and Gutierrez, 2005]. Currently, the Affect Agent is under development.

Central Executive Agent

The Central Executive Agent (CEA) performs cognitive control. It works very closely with the Working Memory System, First-Order Response Agent (reactive control), and Long-Term Memory (Episodic Memory especially) to perform a variety of cognitive activities. CEA is inspired from a human working memory model developed by Baddeley and Hitch as shown in figure 16 [Baddeley and Hitch, 1973]. The phonological loop is responsible for language learning, and the visuo-spatial sketch pad processes visual imagery and information. The central executive, of course, is responsible for controlling the two subsystems.

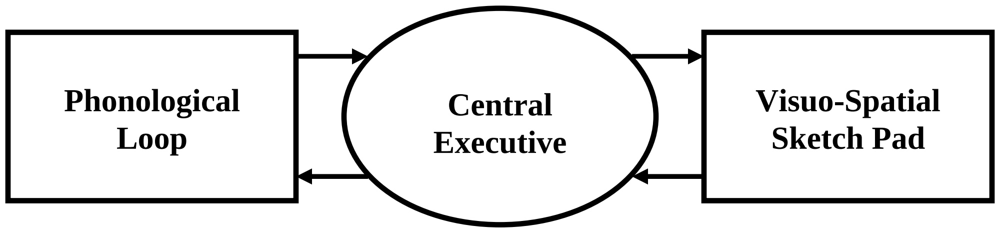


Figure 16: Concept of Working Memory from Baddeley and Hitch [Baddeley and Hitch, 1973]

In its current implementation, CEA is responsible for:

1. Behavior determination by situation
2. Behavior determination from related episodes
3. Maintenance of chunks in working memory during task execution
4. Updating of percept-behavior pairs for FRA

In order for the Self Agent to function properly, CEA and FRA must work together seemlessly [Kawamura et al., 2007a]. FRA and CEA both manipulate chunks in the Working Memory System. However, FRA processes information much faster than CEA, so when ISAC is given a task, FRA will always output chunks to WMS faster than CEA. If FRA places chunks in the working memory that are poor choices, based on the predictions of the Internal Rehearsal System, CEA will carry out two measures. The first measure is to suppress the Activator Agent (AA). This prevents ISAC from performing any further action. The second measure is to search the Episodic Memory for related episodes and find better chunks to place in the working memory. After the superior chunks are found and placed in the Working Memory System by CEA, CEA will unsuppress AA and allow it to initiate the appropriate actions based on the contents of the WMS. This process is further explained in chapter III [Kawamura et al., 2007b].

First-Order Response Agent

The First-Order Response Agent is responsible for the reactive part of ISAC's cognitive architecture. This agent takes the current percepts in ISAC's focus of attention and maps these percepts to a percept-behavior pairs based on ISAC's current intention. The purpose of this agent is to quickly respond to tasks without using a slow deliberative or cognitive process. When a percept-behavior pair is fired given the task, FRA places corresponding chunks into the working memory to be executed by the Activator Agent. This process is further explained in Chapter III [Kawamura et al., 2007a].

Intention Agent

This agent is responsible for determining the intention behind a given task. However, in some cases, the human can have multiple intentions for the given task. The Intention Agent must determine a solution to this conflict by prioritizing the tasks correctly. When the conflict is solved the highest prioritized task will be carried out [Kawamura et al., 2003].

Short-Term Memory

When a percept is detected, the agent posts this information to a short-term memory database system called the Sensory Egosphere (SES) [Peters et al., 2001]. The concept of the Sensory Egosphere was inspired by J. S. Albus [Albus, 1991]. His egosphere concept creates a sphere around the sensor where the sensor is at the center point, and these egospheres can measure position of an object in relation to the sensor, the velocity vector the robot is traveling in, and where the robot is in the world coordinate frame.

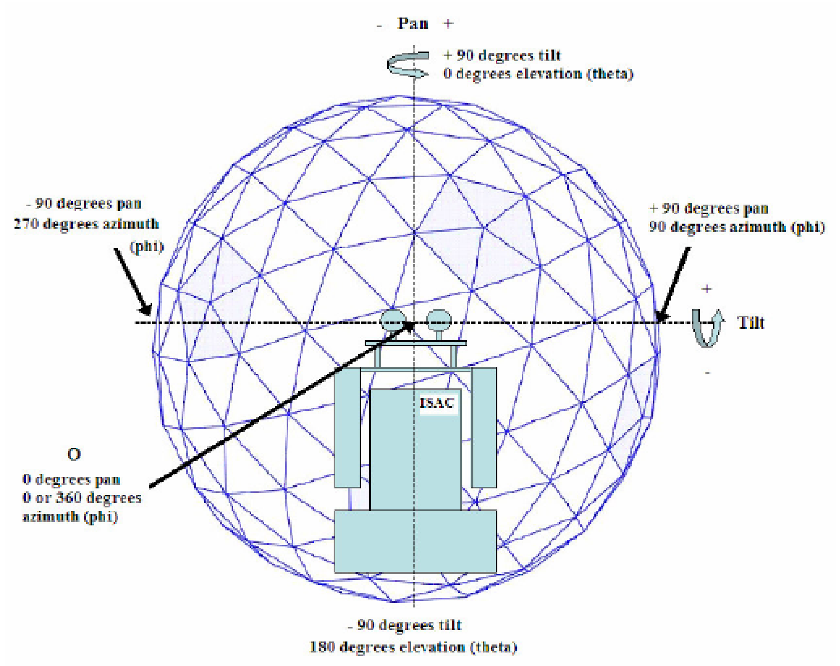


Figure 17: Robot within its Sensory Egosphere [Achim, 2005]

SES is a special case of Albus's egosphere, i.e. a position egosphere, with its center point between ISAC's two cameras as shown in figure 17. This SES has been discretized from a perfect sphere (as with Albus) into a geodesic dome consists of 1962 vertexes each with a corresponding reference node within a MySQL database [Achim, 2005]. Percepts from the environment are assigned to the closest vertex on SES. An example is shown in figure 18.

When a perceptual agent detects a percept, the agent posts this information to a reference node on SES. The percept, known as S in this case, has three critical pieces of information. The first piece is the elevation or θ_s. This value is the angle between the z axis and the percept. The second is φ_s or the azimuth from the x axis, and lastly R_s is the distance from the origin of the Egosphere to the percept. When these values are calculated, the closest node (N_s) is assigned to this percept, and when the percept disappears, SES will delete the percept after a preset time limit expires. Usually, in display situations, only azimuth and elevation are shown on SES. However, for this work, distance is also important, and therefore it is used in the ISAC simulator, as described in chapter III.

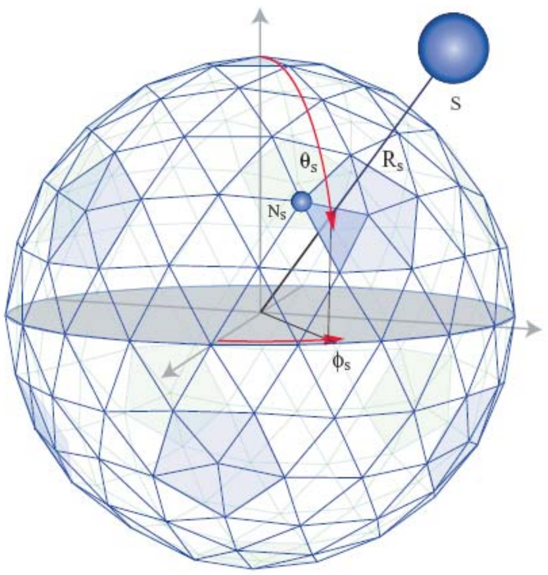


Figure 18: Geodesic Dome SES [Peters et al., 2001]

Long-Term Memory

Long-Term Memory (LTM) contains learned behaviors, facts, and experiences (called episodes) from the past. LTM has three parts: Procedural Memory, Episodic Memory, and Semantic Memory. Procedural memory contains ISAC's learned behaviors. The Episodic Memory contains episodes about specific salient periods of time in the past. Lastly, the Semantic Memory contains facts about percepts that ISAC learned.

Procedural Memory

ISAC's Procedural Memory recalls motion primitives and behaviors allowing ISAC to reach to objects, handshake with guests, and wave its hands in a friendly manner. These behaviors are created by recording motions via tele-operation or motion capture and using a method called Spatio-Temporal ISOMAP on these motions [Jenkins and Mataric, 2003].

Spatio-Temporal ISOMAP is a non-linear technique that uses dimension reduction to extract common spatial and temporal themes between recorded motions (also known as motion exemplars). These themes are generalized and labeled motion primitives. In order to produce learned behaviors, these motion primitives are linked together in series as by the process described in [Erol, 2003]. Figure 19 shows this process in its entirety.

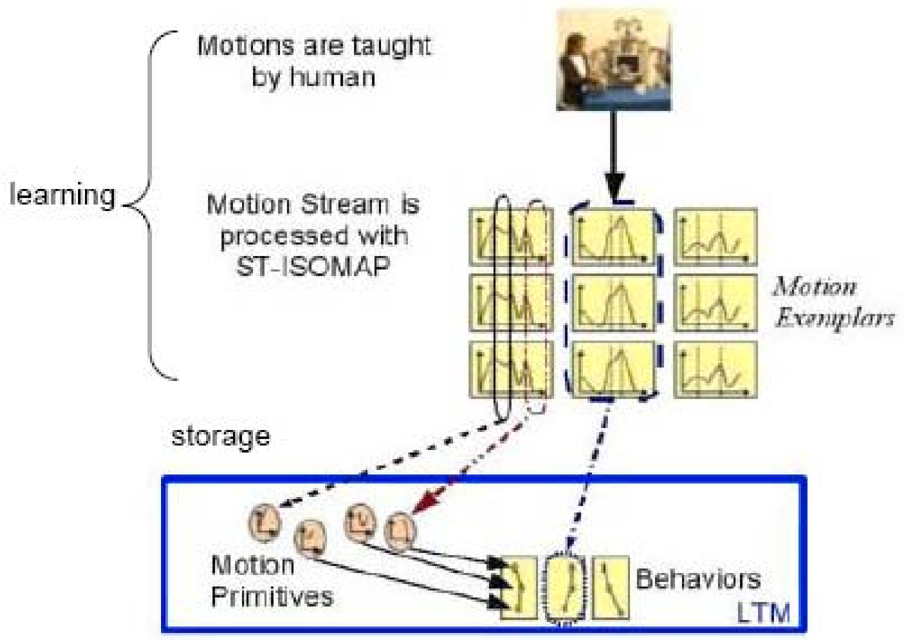

Motions are taught by human
learning
Motion Stream is processed with ST-ISOMAP
Motion Exemplars
storage
Motion Primitives
Behaviors
LTM

Figure 19: Process of Behavior Learning [Erol et al., 2003]

Figure 20 shows how Procedural Memory units are stored. The top of the hierarchy contains the behavior (such as reach or handshake, reach out etc...); Reach out is used as the behavior example in figure 20. The second level contains the motion primitives, and a behavior is linked to this group of associated motion primitives. (Motion primitives 1-3 constitute the reach out behavior in this example). Also, in turn, every motion primitive has a group of raw data files containing the motion exemplars for the primitive. The raw data files have seven fields. The first field is the time the values that were recorded. The next six values are the six joint angles recorded at that set time [Dodd, 2005].

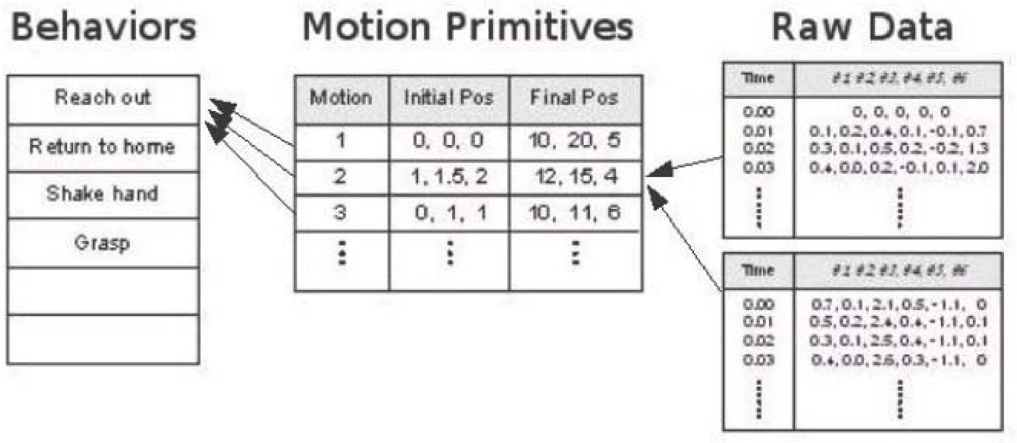


Figure 20: Procedural Memory Example [Dodd., 2005]

When ISAC wishes to create new motions, using the Procedural Memory, a motion interpolation technique is required. This interpolation technique is called *Verbs and Adverbs* [Rose et al., 1998; Spratley, 2006]. In this technique, motions are called verbs and the important parameters are called adverbs. During a motion interpolation a behavior calls up each motion primitive, and each primitive, in turn, calls up its associated motion exemplars. These exemplars are interpolated across to produce a new motion under the banner of its parent behavior. The key to Verb and Adverb interpolation method has to do with motion keytimes. Each raw data file that represents a behavior (like reach for example) has certain key motion exemplars in it. These exemplars may not always take the same amount of time to execute between the raw data files. However, using the kinematic centroid method the keytimes for the reach out motion can be determined from the motion files themselves. Figure 21 shows a mapping between motion times and keytimes [Rose et al., 1998].

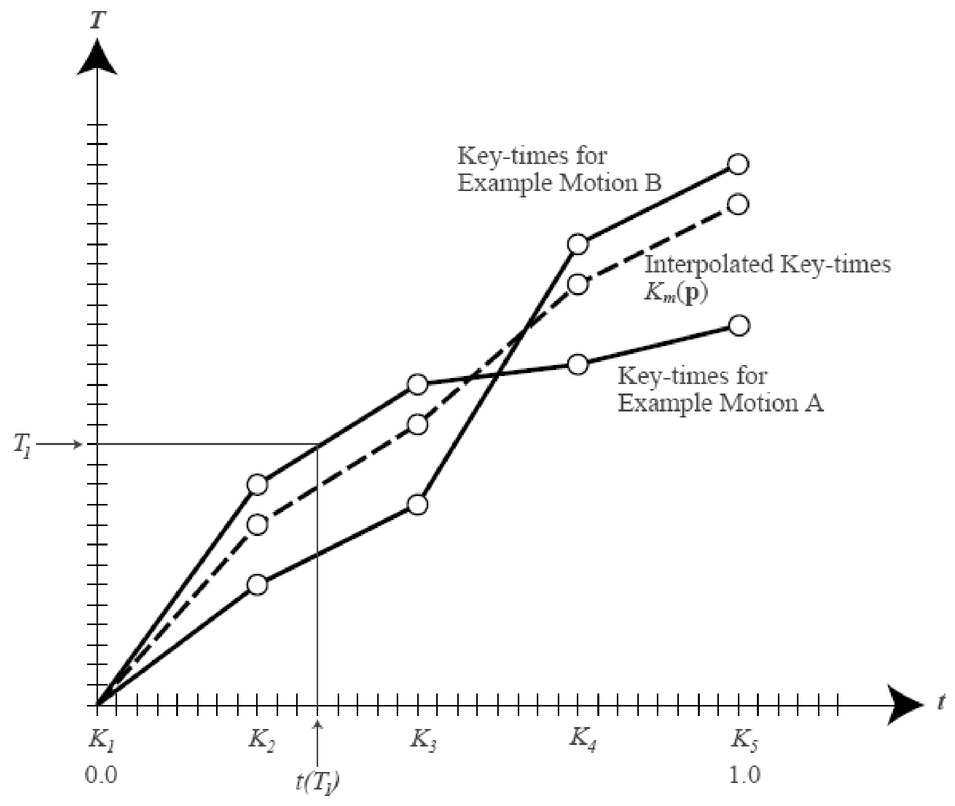


Figure 21: Mapping between Motion Times and KeyTimes [Rose et al., 1998]

Figure 21 shows an example where two motions are recorded that represent one behavior. In this example, five key times are realized via the kinematic centroid method. With these two motions, a third motion is interpolated between the two recorded motions. This interpolation is accomplished by using radial basis functions [Rose et al., 1998].

In most situations, more than two example motions are needed to represent a behavior. For example, ISAC's regular reach motion has six example motions with three adverbs from the SES (azimuth, elevation, and distance). ISAC is able to reach anywhere in its workspace by interpolating between these six motions. The specific set up of ISAC's reach behaviors and Verb/Adverb interpolation are further discussed in chapter III.

Episodic Memory

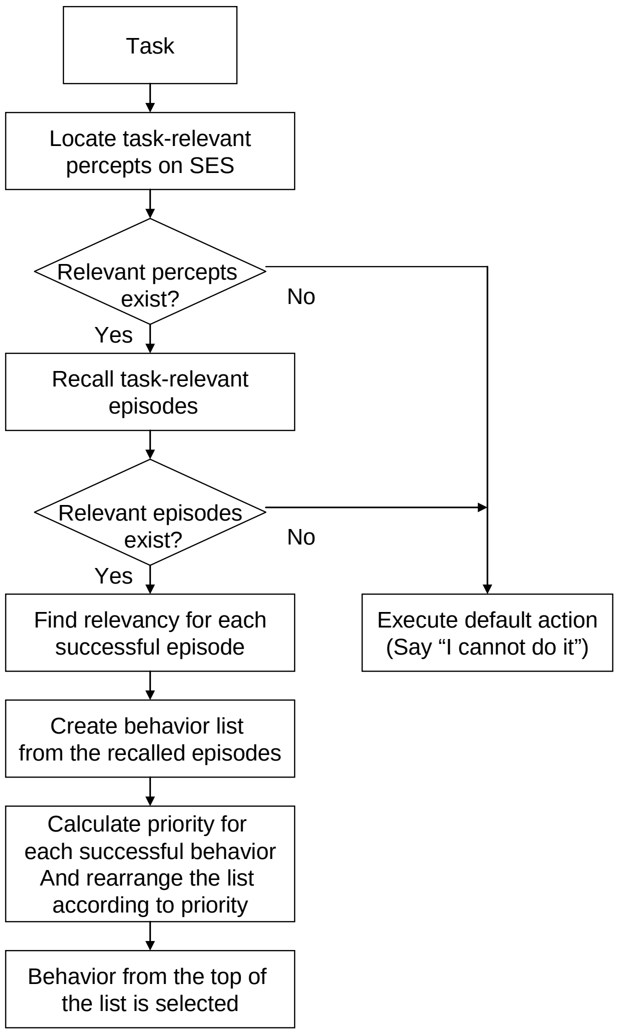
Episodic Memory recalls temporally connected experiences in ISAC's past. These experiences are called episodes. An episode is recorded when ISAC is given a task and the goal remains constant during execution [Dodd, 2005]. Episodic Memory allows ISAC to recall what occurred in the past during a similar task. With this information, ISAC may decide to use the strategy in the episode or not depending on the past success recorded in the episode. Episodic Memory (along with PM and SM) is recorded into a MYSQL database. Table 1 shows an example of Episodic Memory contents. The contents are recorded from the Working Memory System, Procedural Memory, and the SES. Also a timestamp is recorded during the process [Kawamura et al., 2007b].

Table 1: Episodic Memory Contents Example [Kawamura et al., 2007b]

ID	Timestamp	Behavior	Traveling Distance	Percept	SES Node	Task	Result	Internal Salience
1	2007-02-16 14:12:13	reachright	523	lego_toy	1682	[reach lego_toy]	SUCCESS	Emotion VectorID = 1
2	2007-02-16 14:15:45	reachright	613	barney_toy	1684	[reach barney_toy]	SUCCESS	Emotion VectorID = 2
3	2007-02-16 14:44:22	reachright	534	barney_toy	1682	[reach barney_toy]	FAIL	Emotion VectorID = 3
4	2007-02-16 15:00:37	reachright	548	barney_toy	1685	[reach barney_toy]	SUCCESS	Emotion VectorID = 4

Currently, Episodic Memory retrieval is based on keyword matching, but advanced clustering retrieval techniques similar to MIT's CHIP architecture (Comprehensive Human Intelligence and Performance) will be developed for ISAC in the near future [Shrobe et al., 2006].

Every time ISAC is given a task, (like reach to Barney), episodes are pulled from EM based on percepts on the SES and the task itself. If there are no relevant percepts in SES or episodes for the task or tasks, ISAC will say “Sorry, I cannot do it” (see figure 22) [Kawamura et al., 2007b].


Task
Locate task-relevant percepts on SES
Relevant percepts exist?
No
Yes
Recall task-relevant episodes
Relevant episodes exist?
No
Yes
Find relevancy for each successful episode
Execute default action (Say “I cannot do it”)
Create behavior list from the recalled episodes
Calculate priority for each successful behavior And rearrange the list according to priority
Behavior from the top of the list is selected
Figure 22: Behavior Selection using Episodic Memory [Kawamura et al., 2007b]

If ISAC’s EM system does retrieve episodes for the task (labeled T_k), the first thing computed is the expected reward R_k. Reward is mainly used when many conflicting tasks are given to ISAC. Therefore, reward is not shown in figure 22 because this figure deals with one task only. This variable helps ISAC choose which task to perform. Equation 4 describes how the reward is calculated. S_i is the score associated with each episode; this variable has a value of 1 if the episode recorded a success and -1 if it recorded a failure. Lastly, N_k is the number of episodes recalled for the current task [Kawamura et al., 2007b].

$$R_k = \frac{\sum_{i=1}^{N_k} S_i}{N_k} \tag{4}$$

For each episode recalled, its relevancy is computed by the following equation:

$$r = \frac{1}{(d_p + 1) \times (d_b + 1)} \tag{5}$$

In equation 5, d_p represents the number of vertex edges between the percept on the SES and the percept recorded in Episodic Memory. Also, in this equation d_b represents the distance the end effector traveled during the episode. For each behavior (B_j) recalled during the task T_k, there is a list of episodes containing B_j, called E_{Bj}. S_j is the summation of the episodes in E_{Bj} that have a success value of 1. M is another variable which is total summation of episodes in E_{Bj}. These variables form the behavior priority p_j as shown in equation 6.

$$p_j = \frac{S_j}{M} \tag{6}$$

From these equations and the explanation from flow chart in figure 22, the behavior with the highest priority (the one that is shuffled to the top of the list) is the one that is chosen to complete the task [Kawamura et al., 2007b].

Semantic Memory

Semantic Memory (SM) stores facts pertaining to objects perceived by ISAC's sensory systems in the past. SM provides two functions. The first function is storage of declarative facts learned by ISAC. "Nashville is the capital of Tennessee" is an example of a declarative fact. These facts assist ISAC during reasoning and planning stages of the cognitive control system. The second function is storing parameters for perceptual agents such as histograms necessary to recognize certain objects (like Barney for instance) [Dodd, 2005].

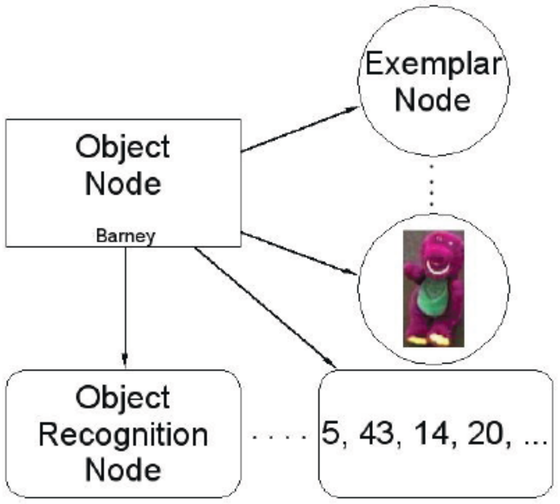


Figure 23: Structure of Semantic Memory [Dodd, 2005]

Currently, this type of LTM is still in the development stage. Will Dodd worked on a Semantic Memory representation; however, it is not currently implemented [Dodd, 2005]. Figure 23 shows an example of Dodd's Semantic Memory structure. SM modes are stored in a MYSQL database in a hierarchical fashion. The object node is at the top, and exemplar nodes and object recognition nodes are attached below. In this example, the object node is of the Barney toy that ISAC plays with. It contains an exemplar node (a picture of Barney) that describes the input that causes this object to be recognized by the perceptual agents, and some object recognition nodes. The object recognition nodes (5, 43, 14, and 20) are codes that describe what types of sensory processes and modes within these processes are used to recognize this percept. For instance, 5 may mean using a process to recognize the color purple [Dodd, 2005].

Working Memory

Working memory is a relatively new memory concept. It is a type of memory found in humans along with some primates. This type of memory resides in the pre-frontal cortex part of the brain, and it is key to cognitive intelligence [Miller et al., 1996]. During task execution, learning, and reasoning, working memory discards distractions, while keeping hold of important task-related information [Baddeley, 1986]. Working memory stores its information as elements called "chunks"; during cognitive processing, these chunks are held in working memory in a completely non-volatile, protected state until they are no longer needed. Working memory has a limited number of chunk slots, some believe this number is 4 [Camperi and Wang, 1998], and others believe it is around 7 with a tolerance of ±2 [Miller, 1956] In either case, this temporary store of chunks is used directly by cognitive processes in the brain, and can be changed or erased very quickly. An example of this concept is remembering a number long enough to dial it. The numbers become chunks in working memory, and once the phone is dialed, the numbers are instantly forgotten [Phillips and Noelle, 2005].

ISAC has a Working Memory System inspired by primate working memory. ISAC's working memory is built upon a neuroscience based C++ toolkit called the Working Memory Toolkit (WMtk). This library is a C++ library specifically tailored for cognitive robotic applications [Skubic et al., 2004]. The toolkit parallels brain chemistry such as the dopamine system. This system in the brain is responsible for determining changes in future reward. WMtk emulates this by using the temporal difference learning algorithm within the toolkit's neural networks [Sutton, 1988]. The working memory toolkit has been used to successfully perform a robotic delayed saccade task as in [Phillips and Noelle, 2006]. Currently, ISAC uses the working memory toolkit to hold task relevant chunks like percepts, behaviors, and sometimes episodes.

CHAPTER III

INTERNAL REHEARSAL

ISAC’s Internal Rehearsal System (IRS) is inspired by Hesslow’s “simulation hypothesis” [Hesslow, 2002]. This idea claims that humans have an “inner world” in which they can internally interact with their environment. The “simulation hypothesis” has three important properties:

Motor Control Simulation – This type of simulation allows the human to think about performing actions without actually performing them. The same part of the brain, the motor cortex, is involved in producing actual motions as well as internally simulated motions.

Perceptual Simulation – This allows humans to imagine an object without the object being present. Perceiving an actual object and internally simulating it excite the same part of the brain, the sensory cortex. In other words, humans do not have to have external stimuli in order to have sensory experiences.

Anticipation of Events – This part of internal simulation is the most important. Hesslow argues that Motor Control Simulation and Perceptual Simulation can be chained together within the brain, multiple times, to anticipate future events by the brain’s cognitive processes. This allows humans to think about the consequences of their actions before the action occurs [Hesslow, 2002].

Internal Rehearsal System (IRS) Concept

ISAC’s Internal Rehearsal System (IRS) is an attempt to use Hesslow’s “simulation hypothesis” within the robot’s cognitive system. IRS allows ISAC to anticipate future consequences given a chosen action. For instance, take the example in figure 24. ISAC is given a task to reach to Barney. The left side of the figure shows the actual robot looking at two percepts, a Lego toy and a Barney doll; the other side shows the ISAC simulator. This simulator is a representation of ISAC’s “inner world”, and it will be further explained later in this chapter. ISAC has three behaviors that it can choose from to accomplish the given task: a *reach right* behavior, *reach right around* behavior, and a *reach left* behavior (labeled 1, 2, and 3 in figure 24 respectively).

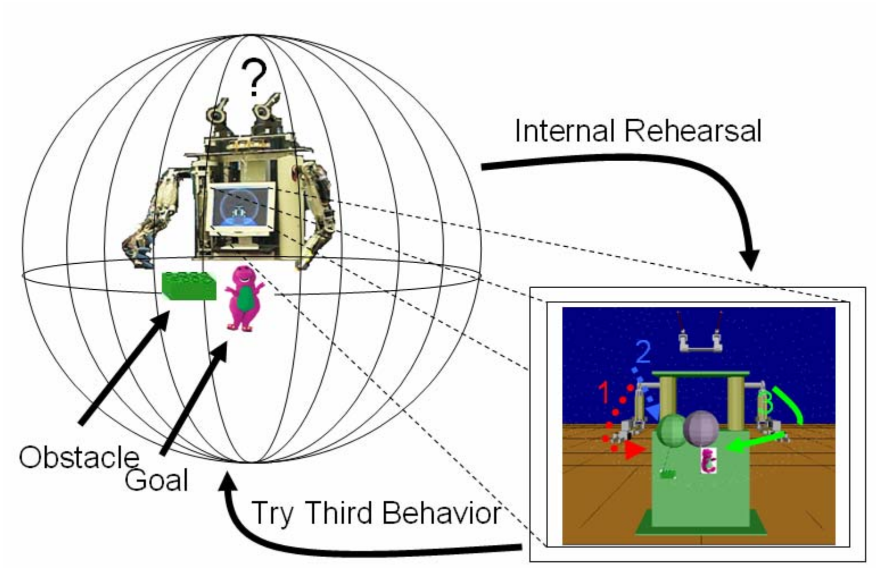


Figure 24: Concept of Internal Rehearsal

First, ISAC tries to reach to the Barney using the *reach right* and *reach right around* behaviors respectively. However, it discovers that these two actions will cause a collision with the obstacle (a Lego toy in this case) by using internal rehearsal. During the internal rehearsal, ISAC discovers that the reach left behavior does not cause a collision with the obstacle, and instead reaches successfully to Barney. Therefore, ISAC decides to reach to Barney with the left arm using a regular reach behavior.

IRS and the Cognitive Cycle

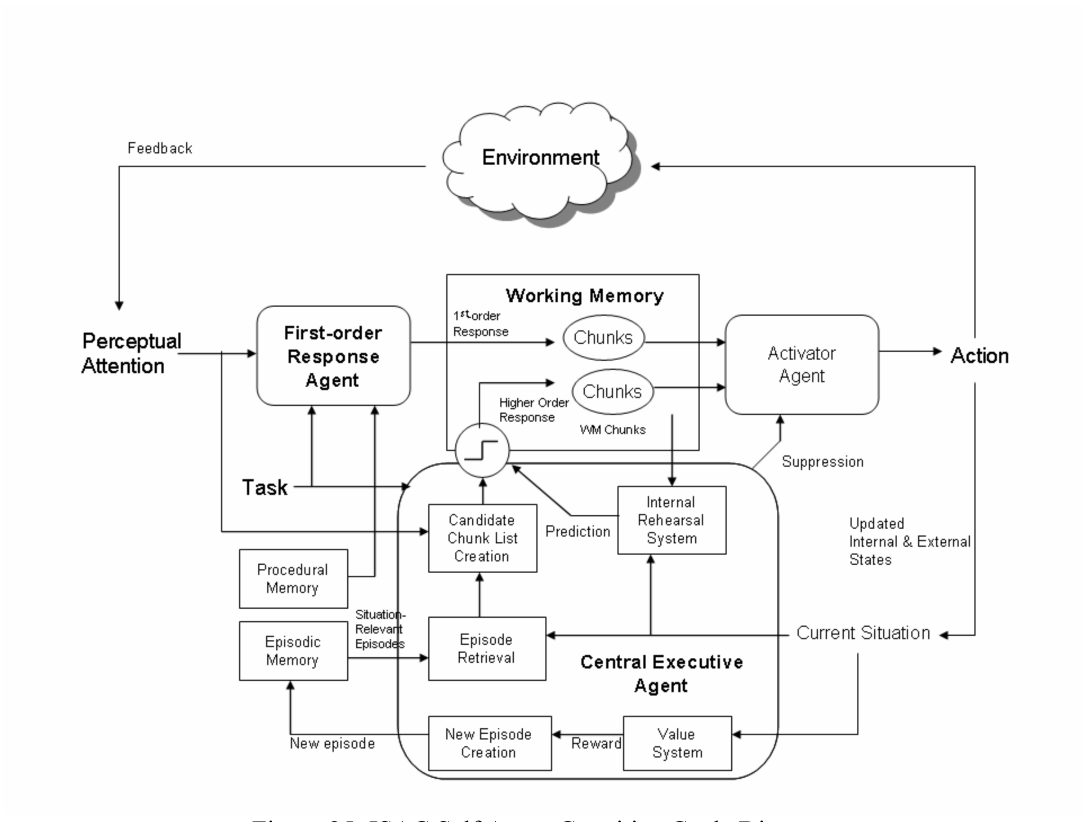


Figure 25: ISAC Self Agent Cognitive Cycle Diagram

The Internal Rehearsal System is a component of the Central Executive Agent, and is a critical piece to ISAC's Self Agent cognitive cycle. Figure 25 shows this cycle. The Self Agent is inspired from Shanahan's cognitive architecture [Shanahan, 2005] based on Baars' Global Workspace concept by utilizing a reactive loop and a cognitive loop [Baars, 1997]. The first loop involves FRA or First-Order Response Agent. It is responsible for generating reactive and routine responses given stimuli from the perceptual system of the robot and the current task. FRA matches percepts to behaviors using symbolic if-then rules and passes these percept-behavior pairs to the working memory system as chunks. The Central Executive Agent can be considered to be the cognitive control system. This agent is responsible for switching the working memory chunks if they are not suitable for successful task execution. In other words, CEA behaves similarly to the Higher-Order Loop in Shanahan's work [Shanahan, 2005].

The Internal Rehearsal System allows CEA to internally simulate the actions using the working memory chunks. IRS takes the current situation (specifically the external state of the robot) and the working memory chunks as input, and it produces a prediction about the success or failure of the chosen working memory chunks. This prediction is a mechanism used by CEA for suppression determination. For instance, if IRS determines that the given working memory chunks will fail to complete the goal, CEA will suppress the Activator Agent and change the WMS chunks based on relevant episodes. This means that the Activator Agent and Internal Rehearsal System process at the same time when working memory chunks appear or are changed. IRS must be much faster than the Activator Agent in order for CEA to successfully suppress it. When the Activator Agent is suppressed, the Central Executive Agent will replace the contents in the Working Memory System.

When a task is given to ISAC, CEA generates a candidate behavior list from the Episodic Memory contents; at the same time, FRA places chunks, i.e. a percept-behavior pair, into Working Memory based on the learned percept-behavior pairs. When chunks appear or are changed in the WMS, IRS processes them. If IRS returns a prediction that CEA finds acceptable, CEA will allow the Activator Agent to perform the behavior without interruption. However, if CEA receives a prediction that it does not like, CEA will suppress the Activator Agent and will remove this particular behavior from its list. Next, CEA will try the next behavior from its list and translate this, along with the accompanying percept, into the working memory chunks. This loop will continue until a suitable behavior is found or until CEA's list is exhausted. When this happens, the Activator Agent will simply perform the default action. Currently, this default action is ISAC saying "Sorry, I cannot do it". This process is illustrated in more detail in figure 26 [Kawamura et al., 2007b].

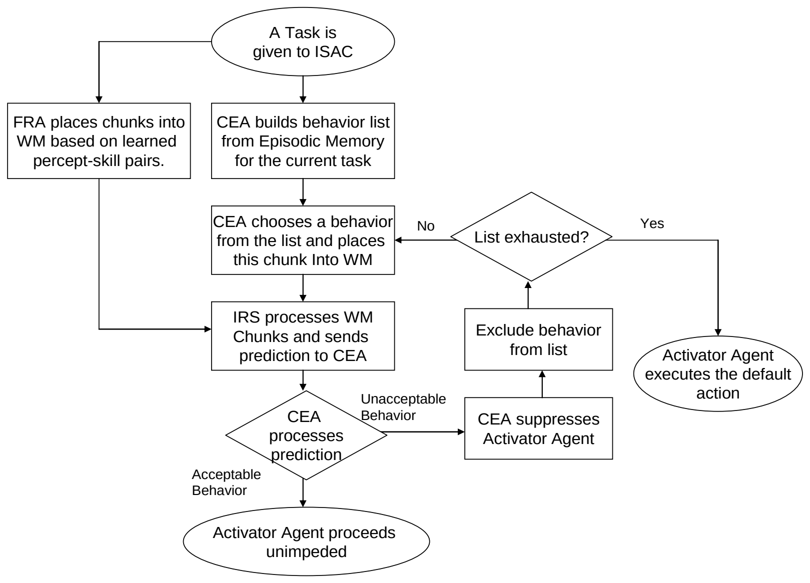

A Task is given to ISAC
FRA places chunks into WM based on learned percept-skill pairs.
CEA builds behavior list from Episodic Memory for the current task
CEA chooses a behavior from the list and places this chunk Into WM
No
List exhausted?
Yes
IRS processes WM Chunks and sends prediction to CEA
Exclude behavior from list
Activator Agent executes the default action
CEA processes prediction
Unacceptable Behavior
CEA suppresses Activator Agent
Acceptable Behavior
Activator Agent proceeds unimpeded

Figure 26: Flow Chart of CEA Execution involving Internal Rehearsal [Kawamura et al., 2007b]

In order to produce a prediction about how the behavior will be positive or negative, IRS uses collision detection methods similar to [Charoenseang, 1999]. Charoenseang's work used virtual spheres around the end effector, wrist, and elbow of both the right and left arms and each percept in ISAC's environment. These virtual spheres produced a virtual force feedback upon contact with another sphere. This prevented damage to ISAC's arms by preventing them from colliding with percepts or each other. Internal Rehearsal uses this concept somewhat differently. The percept is seen as a sphere where the area inside of the sphere is seen as a collision domain. This will be explained more in the next section.

ISAC Simulator and Collision Detection

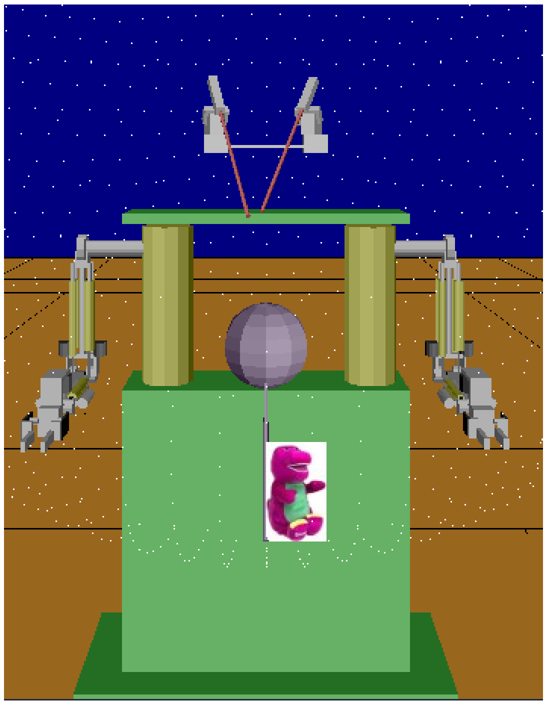


Figure 27: ISAC Simulator

IRS also makes use of a program called ISAC Simulator. This program, shown in figure 27 is an OpenGL representation of ISAC. Palis Ratanaswasd, inspired by Charoensang's work, programmed the original version of this simulator [Charoenseang, 1999; Ratanaswasd et al., 2005]. This program displays what the Internal Rehearsal System is currently doing along with the percepts on SES. Figure 27 shows ISAC internally perceiving a Barney percept on SES. The purple sphere is the Barney collision sphere that is located in Cartesian coordinates in the simulator.

For each percept, SES stores its location of information in terms of the azimuth (Φ_s), elevation (θ_s), and distance (R_s) with respect to the head as discussed in Chapter II. IRS translates this information on SES into Cartesian coordinates by the following functions.

$$D_{2R} = \frac{\pi}{180} \tag{7}$$

$$x_S = 1000 \cdot R_S \cdot \sin(D_{2R} \cdot \theta_S) \cdot \cos(D_{2R} \cdot \phi_S) \tag{8}$$

$$y_S = 1000 \cdot R_S \cdot \sin(D_{2R} \cdot \theta_S) \cdot \sin(D_{2R} \cdot \phi_S) \tag{9}$$

$$z_S = 1000 \cdot R_S \cdot \sin(D_{2R} \cdot \theta_S) \tag{10}$$

These Cartesian coordinates are in terms of the SES's origin point. IRS checks collisions with the virtual arm's coordinate system. The virtual arms coordinate system is calibrated in the same manner as the actual arm's coordinate system. Therefore, another set of equations are required to put the percept in the IRS Cartesian coordinates. O_S is the origin of SES in the z axis, and O_A is the origin of both the right and left virtual arm in the z axis.

$$O_S = 1655 \tag{11}$$

$$O_A = 1520 \tag{12}$$

$$x_{IRS} = y_S \tag{13}$$

$$y_{IRS} = -x_S \tag{14}$$

$$z_{IRS} = z + (O_S - O_A) \tag{15}$$

During the rehearsal, either the right or the left arm will perform a behavior dictated by the contents of working memory. If the WMS contains chunks "ReachRight1" and "Barney", ISAC will internally rehearse a reach with the virtual right arm using IRS and the arm motion will appear on the simulator.

When the Internal Rehearsal System performs a behavior, IRS uses the same Verbs and Adverbs technique as the Arm Agents. Both the real arm and the virtual arm interpolate a similar trajectory towards the goal percept. However, IRS uses a stepping approach to speed up the process. This will be further explained in the Verbs and Adverbs section later.

Internal Rehearsal System's collision detection is slightly different from Charoensang's approach. Instead of having virtual spheres around the joints and end effector of each arm, the percepts located on the SES are seen as spheres. This spherical object representation has been used with mobile robotic navigation before where the obstacle is revolved into a sphere [Vikenmark and Minguez, 2006]. In order to detect a collision with a percept, the virtual arms within the Internal Rehearsal System must have a method for accomplishing this task. The method is shown in figure 28. Both arms have seven points called collision points; they are located on the shoulder, bicep, elbow, upper and lower forearms, wrist, and end effector of each virtual arm.

These seven points are more concentrated on the lower part of the virtual arm because this area tends to have more collisions. The upper forearm collision points do not tend to have many collisions, but they are used to show the trajectory of the upper arm during an internal rehearsal. This gives context to the other four points in the experimental result graphs. Also, the end effector collision point is not located at the tip of the end effector. This is for two reasons. 1) ISAC's actual end effectors are not nearly as large as the simulator's end effectors. 2) The collision spheres used in the experiments have a large enough radius to easily detect this collision point if the tip of the end effector touches a percept. The collision points are determined by finding the Cartesian coordinates of each point via forward kinematics. More information about ISAC's forward kinematics functions can be found in [Schroder, 2003]. When any of the seven collision point enters a percept collision sphere, a collision has occurred and a prediction is sent back to CEA.

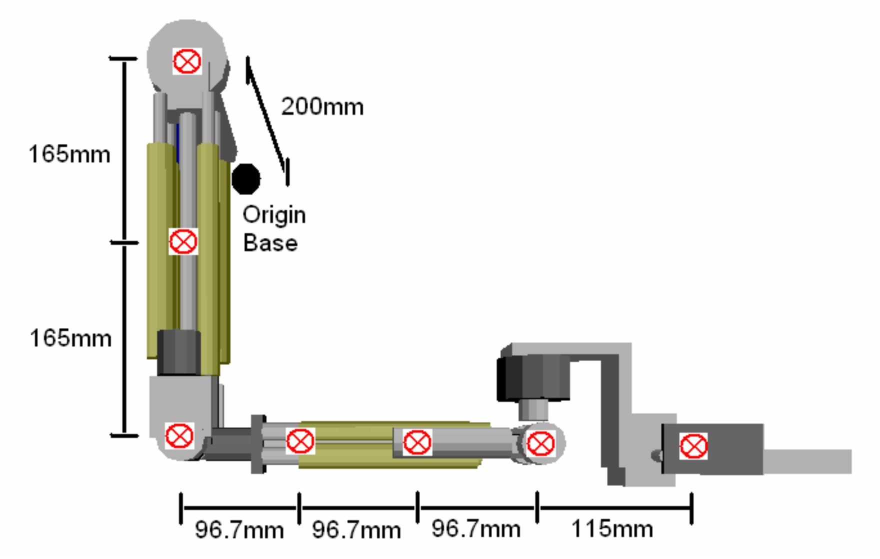


Figure 28: Right Virtual Arm within IRS

Output of IRS Prediction

When IRS gives a prediction to CEA, IRS sends the following information as a text string with the following variables (table 2):

Table 2: Arguments in IRS Prediction

Prediction Order	Variables	Type
1	Joint Step	Integer
2	Joint Size	Integer
3	End Effector Distance	Double
4	Collision Percept	String
5	Collision Switches	Vector of Booleans

The joint step value is an integer that expresses the step in the virtual arm trajectory where the collision occurred. The joint size value is the total number of steps in the trajectory. These two values together tell CEA where in the trajectory the collision happened, near the beginning or near the end. The End Effector Distance is the traveled distance of the end effector collision point during the internal rehearsal. This tells CEA how much distance the end effector will travel during the internally rehearsed behavior. This value is also known as d_b in the Episodic Memory section in chapter II. The Collision Percept is the collision sphere entered during the rehearsal; this is a string such as “Barney_toy” or “Lego_toy”. The last part of this string is a vector containing the collision switches. This vector contains seven booleans each representing a collision point. The collision point order is shown in table 3.

Table 3: Order of the Collision Point Vector

Vector Order	Collision Point
1	End Effector
2	Wrist
3	Upper Forearm
4	Lower Forearm
5	Elbow
6	Bicep
7	Shoulder

IRS and Verbs and Adverbs

In order to evaluate the performance of IRS, the Right Arm Agent and the right virtual arm in IRS were used to produce two different behaviors for the experiments in chapter IV. These two behaviors are *reach* and *reach around*. *Reach* is simply a direct reaching motion from the rest position to the object and back; this behavior is also known as *regular reach*. *Reach* *around* involves ISAC trying to reaching to the object from above. This behavior is useful when an object is present that prevents ISAC from reaching to the goal object directly. Each behavior (verb) has six examples (or files with the motion exemplars) to interpolate from, and each example has three adverbs associated with it (azimuth, elevation, and distance). The information for both are shown below in tables 4 and 5:

Table 4: Motion Examples for "Reach" Verb

Motion	Azimuth	Elevation	Distance
1	0.02	128.82	0.5270
2	11.06	135.62	0.6638
3	10.22	134.54	0.8627
4	-10.6	134.10	0.8991
5	-28.24	137.06	1.0473
6	-7.90	133.36	0.9953

Table 5: Motion Examples for "Reach Around" Verb

Motion	Azimuth	Elevation	Distance
1	-3.716	146.19	0.9434
2	-34.15	143.60	0.9659
3	22.98	129.39	0.8320
4	-6.12	131.97	0.6360
5	-9.64	129.15	0.8604
6	16.56	141.75	0.9282

For each motion example in both tables 4 and 5, the Barney doll was placed in front of ISAC. ISAC's cameras would saccade to the object and place its location on SES. The three SES values, (azimuth, elevation, and distance) were then used as the three adverbs for the motion. The example motion was recorded by teleoperation and stored in ISAC's procedural memory. As seen in both tables, six example motions, with six different sets of adverbs, were recorded for each behavior (*reach* and *reach around*). This allows ISAC to smoothly interpolate a reach anywhere in ISAC's workspace.

Verbs and Adverbs Algorithm

The Verbs and Adverbs algorithm depends on determining the location of example motion's keytimes. A keytime is defined as, "a critical structural element in the motion's trajectory that is common to all motions of that type" [Spratley, 2006]. For each example motion, a number of keytimes will appear by using the Kinematic Centroid Method [Rose et al., 1998].

Table 6: Verb and Adverb Symbols [Spratley, 2006]

Symbol Name	Description
NumDOF	Number of Degrees of Freedom in the trajectory
NumAdverbs	Number of Adverbs that parameterize the motion
NumExamples	The number of example motions
NumPoints	Total number of data points in the trajectory
NumKeyTimes	Number of KeyTimes
k	$k \in Z^{nonneg} < NumKeyTimes$, the KeyTime Number
KT	point in the resample trajectory corresponding to a keytime
A	Adverb matrix, independent variable in a least squares fitting
B	Trajectory matrix, dependent variable in a least squares fitting
$\bar{B}$	error matrix after least squares fitting
$\bar{B}$	least squares approximation of trajectory
x	matrix of linear coefficients
r	matrix of radial-basis function coefficients
D	matrix of gaussians
tt	time index

Between all example motions of a certain behavior, the keytimes may be slightly different from each other, but the number of keytimes, and the place in the example motion stream where they occur will be the same [Spratley, 2006].

Each processed example motion will have unevenly spaced keytimes. Therefore, the motion files are normalized so that the key times are evenly spaced from one another. This normalization uses linear interpolation (a least squares technique) between the keytimes. This approach not only allows the algorithm to interpolate between the example motions, but to extrapolate as well. The equation for this least squares technique is shown in equation 16 where x is a linear coefficient matrix with NumAdverbs+1 rows and NumDOF columns, A is the adverb matrix (NumExamples rows and NumAdverbs+1 columns), and B is the dependant trajectory matrix (NumExamples rows and NumDOF columns): [Spratley, 2006]

$$Ax[tt] = B[tt] \tag{16}$$

$$Ax[tt] = \widetilde{B}[tt] \tag{17}$$

$$\overline{B}[tt] = \widetilde{B}[tt] - B[tt] \tag{18}$$

By finding x[tt], the linearized approximation of B[tt] can be found (equation 16). The results of equations 16 and 17 are used to find the trajectory error matrix (calculated in equation 18). This residual error allows the Verb/Adverb algorithm to extrapolate beyond the confines of the example motions; it basically captures the nonlinear properties of the recorded motions [Spratley, 2006].

The residual error matrix is used to define the radial basis function coefficients as seen in equation 19.

$$r[tt] = D^{-1}\overline{B}[tt] \tag{19}$$

The other matrix, D, is known as the matrix of Gaussians; it holds the radial basis functions for the radial basis function matrix computation. The D matrix Gaussians are computed via the distance from the interpolated motion's adverbs to the existing example motions's adverbs. More information about this process can be found in [Spratley, 2006].

When all of the matrices have been successfully computed, a new B[tt] (trajectory) matrix can be computed as seen in equation 20. This equation creates the interpolated trajectory for the new Verbs and Adverbs [Spratley, 2006].

$$B[tt] = Dr[tt] + Ax[tt] \tag{20}$$

IRS Stepping

ISAC's Arm Agents along with IRS have two phases during a Verb and Adverb motion. The first phases is the trajectory generation phase. This phase involves the Verb/Adverb algorithm interpolating an arm joint trajectory for the arm to follow. The second phase is the movement phase. During this second phase, the system focuses on moving the arm itself on this interpolated trajectory. There are two differences between the Arm Agents and IRS.

1. During the movement phase IRS checks to see if a collision has occurred for each trajectory joint step. However, the Arm Agent only focuses on movement.
2. During the interpolation phase, IRS is able to perform Verb/Adverb stepping to produce a trajectory similar to and faster than the Arm Agents' trajectory.

IRS uses Verb/Adverb stepping to speed up both phases of this process. Stepping involves taking each of the example trajectories and skipping over some of the recorded joint values. The current convention is to store example trajectories as seven columns into an example trajectory file recording the time for the first value and the six joint angles of the arm as the rest. Table 7 shows a very short verb example file.

Table 7: Short Verb Example File

Time	Theta1	Theta2	Theta3	Theta4	Theta5	Theta6
0.00	-0.13	1.72	-1.53	1.78	-0.83	0.27
0.05	-0.14	1.72	-1.53	1.78	-0.83	0.27
0.10	-0.15	1.72	-1.54	1.78	-0.83	0.27
0.15	-0.16	1.72	-1.54	1.78	-0.83	0.27
0.20	-0.17	1.72	-1.55	1.78	-0.83	0.27
0.25	-0.18	1.72	-1.55	1.78	-0.83	0.27
...	...	...	...	...	...	...

Table 8: Verb Example File (stepping of 2)

Time	Theta1	Theta2	Theta3	Theta4	Theta5	Theta6
0.00	-0.13	1.72	-1.53	1.78	-0.83	0.27
0.05	-0.15	1.72	-1.54	1.78	-0.83	0.27
0.10	-0.17	1.72	-1.55	1.78	-0.83	0.27
...	...	...	...	...	...	...

For example, table 8 shows an example file stepped by 2. In a stepping of 2, one out of every two joint steps is processed by ISAC's Verb and Adverb algorithm. The time index is also altered. The third row of the original example file becomes the second row of the new stepped example file and so on. This process continues for the entire file.

The keytimes in stepped files are the same, the only difference is that the stepped examples are smaller and faster to interpolate. This creates a duel advantage. IRS is able to interpolate faster than the Arm Agent and is able to move faster (in simulation) while checking for collisions. ISAC's cognitive system will work correctly if IRS is able to complete its two phases before the Arm Agent finishes its first phase. Otherwise, the arm will begin to move while CEA is still processing information. Currently, IRS fully works with steppings of 1 through 5. A stepping of 1 uses all of the joint steps in each recorded example trajectory. However, a stepping of five uses only one joint step out of five in each trajectory.

CHAPTER IV

IRS EXPERIMENTS

The purpose of the first experiment is to determine how the stepping number will affect IRS performance, and to determine if IRS can read chunks from memory correctly. The purpose of the second experiment is to show how CEA uses IRS for task execution.

Both experiments use one or both of the objects shown in figure 29. The object on the left is the Barney doll, and the object on the right is the Lego toy. The Barney has a height of 240 mm, a width of 110 mm, and a depth of 130 mm (no tail). The Lego toy has a height and width of 190 mm, but it has a diagonal width (corner to corner) of 240mm. It has a depth of 40 mm. I decided to use the largest dimension (height in this case) to represent the size of the sphere. These two objects are similarly sized. So a sphere of equal sizes should be sufficient for both. Originally, a sphere with a radius of 125 mm was tried. However, this radius size caused the spheres to overlap. This is because the black stands on which the objects rest have a base diameter of 240mm (or a radius of 120mm). This overlap can confuse the Central Executive Agent because IRS can think that it is colliding into two objects at the same time. So in order to prevent problems with overlap, 115mm radius was chosen for the collision spheres.

Figure 29: Picture of Barney and Lego Toys

Experiment 1

As stated previously, the purpose of the first experiment is to determine how the stepping number will affect IRS performance, and to determine if IRS can read chunks from memory correctly. In this experiment, Barney will be presented before ISAC on a stationary stand with no obstacles present. During this time, ISAC will be asked to reach to the Barney. The experimental set up is seen in figure 30. The following assumptions hold during this experiment:

1. FRA has some knowledge about the task, and will invoke a routine response based on previous knowledge of the task. The routine response will be to perform a regular right reach to the object.
2. The object (Barney doll) will be stationary during the experiment.
3. The object is positioned to where ISAC can reach to it.

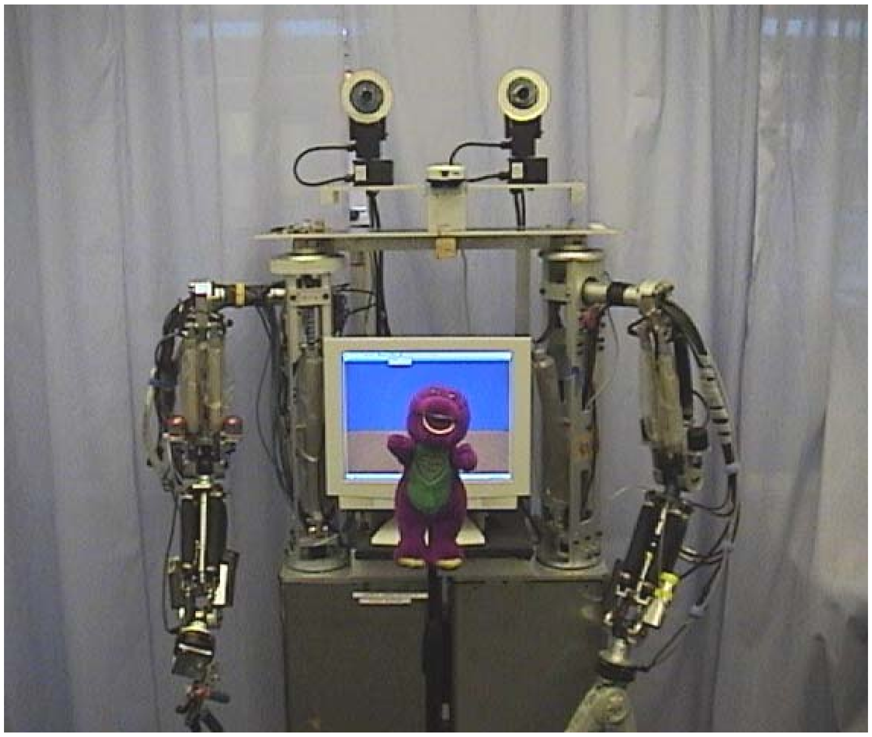


Figure 30: Setup for Experiment 1

ISAC will have two right reach behaviors and one left reach behavior in its repertoire. When the task is presented, FRA will post chunks to WM stating “ReachRight1” and “Barney” meaning perform a regular reach with the right arm to the Barney doll. IRS and the Activator Agent will both process these chunks at the same time, and depending on IRS’s prediction, CEA will or will not replace these chunks. If IRS produces the correct prediction, CEA will not get involved and will not tamper with the FRA’s chunks. This process is shown in figure 31; the most important agents and modules are highlighted in the dashed-line area.

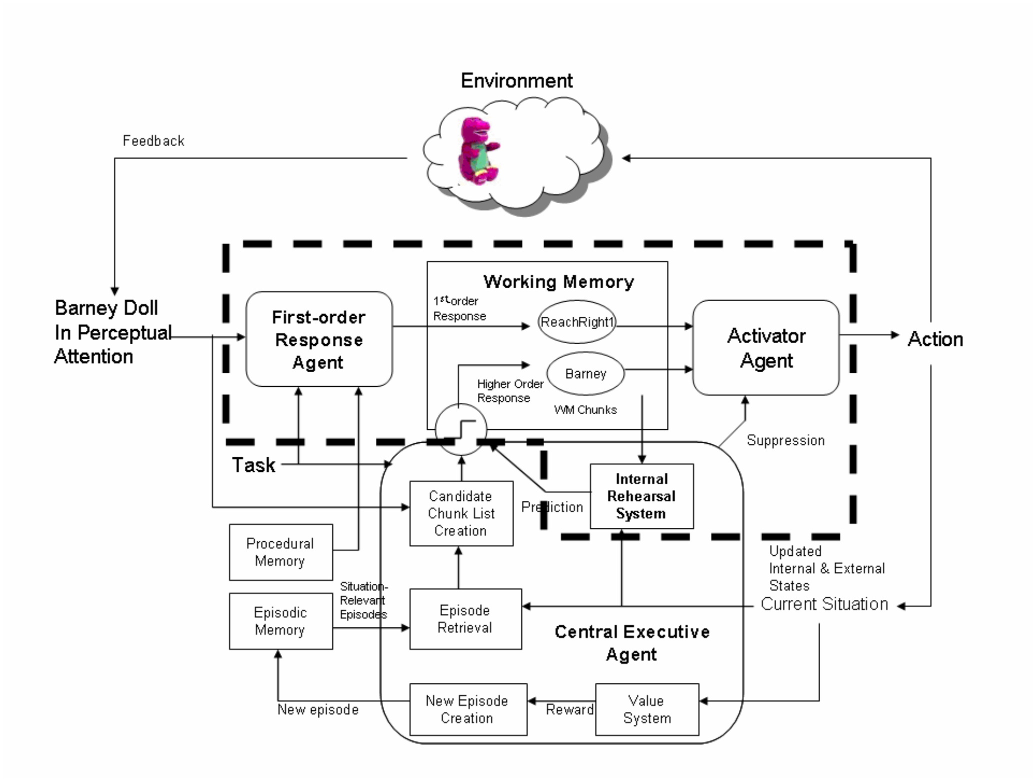


Figure 31: Self Agent Cognitive Cycle for Experiment 1

Also, in this experiment, steppings 1 through 5 are used to determine which causes the fastest and most reliable prediction. Stepping 1 will obviously be the slowest, but it will also be the most reliable. The stepping of 5 will be the fastest, but it could miss the percept entirely.

Experiment 2

As stated previously, the purpose of the second experiment is to show how CEA uses IRS for task execution. Experiment 2 involves an obstacle, namely a Lego toy along with the target object, Barney. As with Experiment 1, both percepts are perched on top of stands a seen in figure 32.

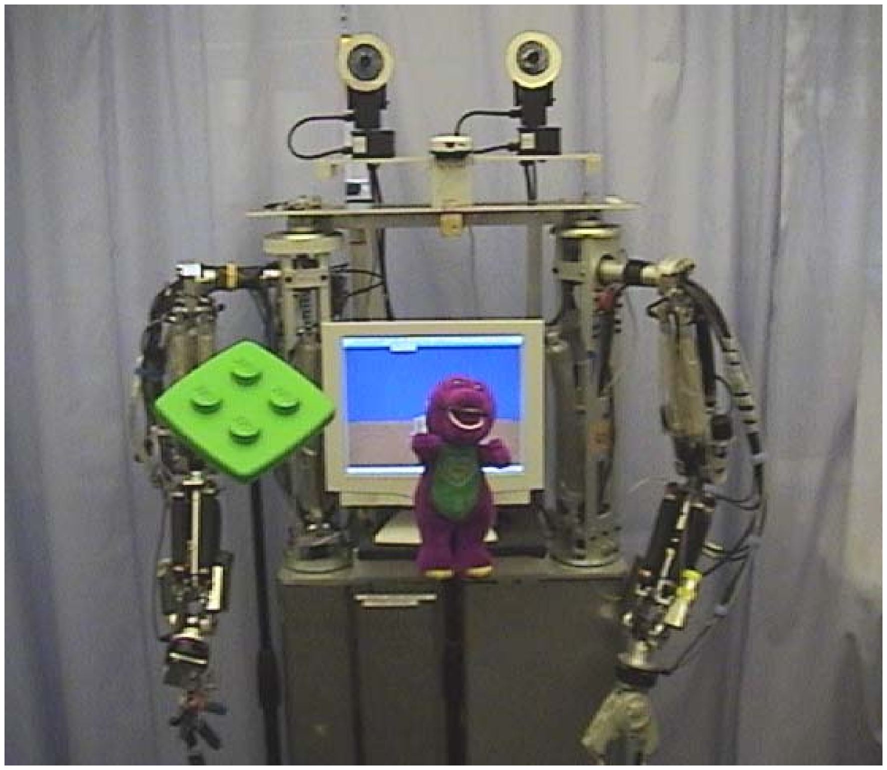


Figure 32: Setup for Experiment 2

During this experiment, ISAC will be given a task to reach to Barney. CEA, by using IRS and the Episodic Memory (EM), must find the proper behavior to accomplish this task. EM will retrieve episodes that are most likely to perform the task successfully (see chapter III). The Lego toy will cause the Internal Rehearsal System to find collisions with the Lego toy with both the regular right reach and the right reach around, but the left regular reach behavior will be successful. For this experiment, the following assumptions hold:

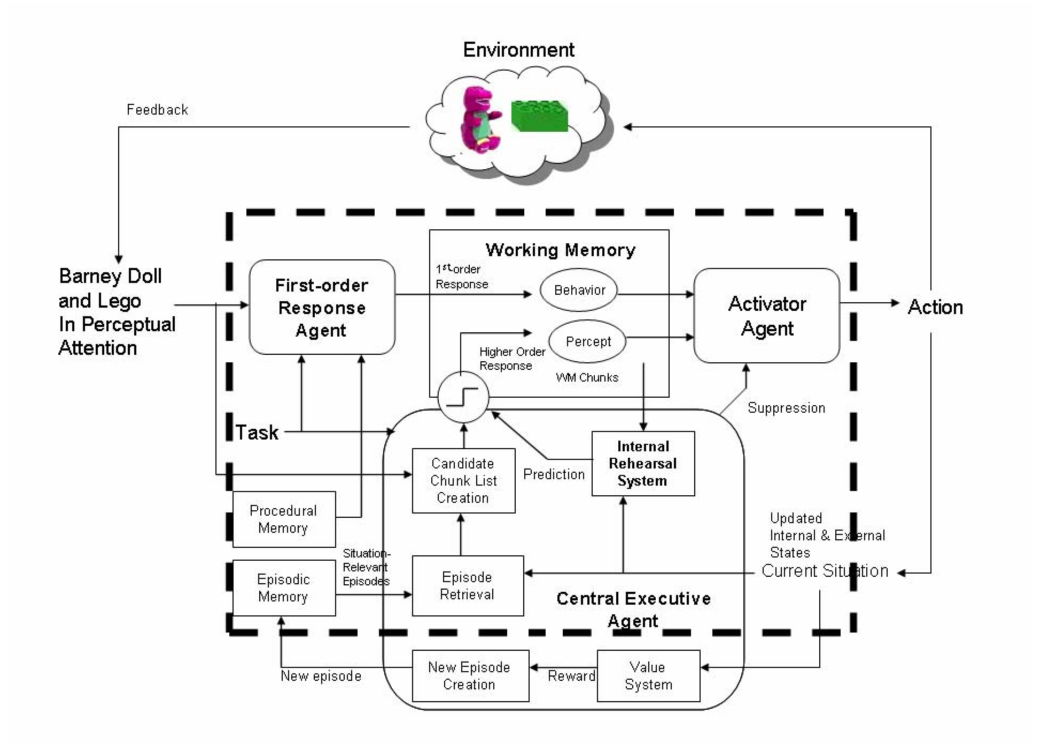


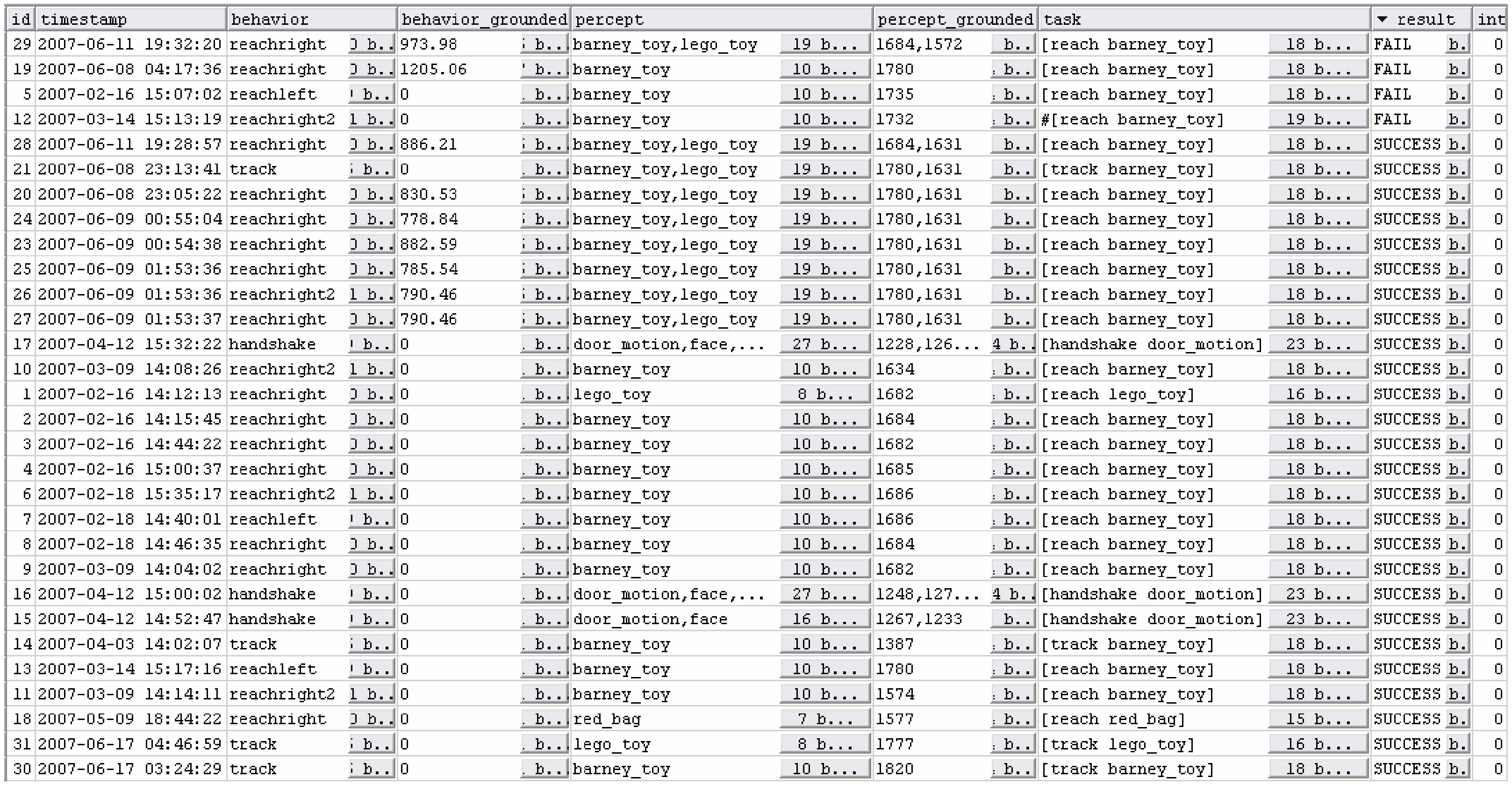
Figure 33: Self Agent Cognitive Cycle for Experiment 2

1. FRA has some knowledge about the task, and will invoke a routine response based on previous knowledge of the task. The routine response will be to perform a regular right reach to the object.
2. ISAC's Episodic Memory is bootstrapped with the necessary experience to complete the task at hand (reaching in this case). The contents of the Episodic Memory can be seen in figure 34.
    a. ISAC has the most experience using the regular reach right behavior to successfully perform a reaching task.
    b. Reach right around has the second most experience.
    c. Reach left has the least experience in Episodic Memory and is used last.
3. The objects (Barney doll and Lego toy) will be stationary during the experiment.
4. The target object is positioned to where ISAC can reach to it.

FRA will place chunks into the working memory, “ReachRight1” and “Barney”. During this time, CEA will be processing a behavior list from relevant episodes in the Episodic Memory, and IRS along with the Right Arm Agent will process FRA chunks. IRS will discover a collision with the Lego toy i.e. not satisfying the goal. CEA then will remove this behavior from its behavior list, suppress the Activator Agent, and try the next best behavior -- reach right around. CEA will load “ReachRight2” and “Barney” into working memory. This behavior will also produce a poor prediction by colliding with the Lego toy in IRS. CEA will remove this behavior from its list and try the last behavior, reaching left. This behavior will be translated into chunks “ReachLeft1” and “Barney”; they will be placed into WMS. IRS will try this behavior and find that it will successfully reach to the Barney toy. CEA will thus not suppress the Activator Agent, and ISAC will reach to the Barney with the left arm. The important agents and modules for this process are highlighted in figure 33.

Figure 34 shows the contents of the Episodic Memory. This figure contains a table directly from the EM MYSQL database. The database figure follows the same format as table 1 in the Episodic Memory section of chapter II. The only difference is that the behavior_grounded field is synonymous with travel and the percept_grounded field is synonymous with SES Node. This database shows that there are many more successful reach right behaviors than reach right around or reach left behaviors. This means that ISAC has the most experience using right hand behaviors than left hand behaviors. And thus, the assumptions for this experiment hold true.

As with Experiment 1, the steppings 1 through 5 will be used. This time the steppings will be more important because if IRS is too slow, the Activator Agent will move the arm before CEA can suppress it. Also, if IRS is inaccurate in its predictions, ISAC could exhibit strange behavior by reaching to the object with the obstacle present.



id	timestamp	behavior		behavior_grounded		percept		percept_grounded		task		▼ result		int
29	2007-06-11 19:32:20	reachright	b..	973.98	b..	barney_toy,lego_toy	19 b...	1684,1572	b..	[reach barney_toy]	18 b...	FAIL	b.	0
19	2007-06-08 04:17:36	reachright	b..	1205.06	b..	barney_toy	10 b...	1780	b..	[reach barney_toy]	18 b...	FAIL	b.	0
5	2007-02-16 15:07:02	reachleft	b..	0	b..	barney_toy	10 b...	1735	b..	[reach barney_toy]	18 b...	FAIL	b.	0
12	2007-03-14 15:13:19	reachright2	b..	0	b..	barney_toy	10 b...	1732	b..	#[reach barney_toy]	19 b...	FAIL	b.	0
28	2007-06-11 19:28:57	reachright	b..	886.21	b..	barney_toy,lego_toy	19 b...	1684,1631	b..	[reach barney_toy]	18 b...	SUCCESS	b.	0
21	2007-06-08 23:13:41	track	b..	0	b..	barney_toy,lego_toy	19 b...	1780,1631	b..	[track barney_toy]	18 b...	SUCCESS	b.	0
20	2007-06-08 23:05:22	reachright	b..	830.53	b..	barney_toy,lego_toy	19 b...	1780,1631	b..	[reach barney_toy]	18 b...	SUCCESS	b.	0
24	2007-06-09 00:55:04	reachright	b..	778.84	b..	barney_toy,lego_toy	19 b...	1780,1631	b..	[reach barney_toy]	18 b...	SUCCESS	b.	0
23	2007-06-09 00:54:38	reachright	b..	882.59	b..	barney_toy,lego_toy	19 b...	1780,1631	b..	[reach barney_toy]	18 b...	SUCCESS	b.	0
25	2007-06-09 01:53:36	reachright	b..	785.54	b..	barney_toy,lego_toy	19 b...	1780,1631	b..	[reach barney_toy]	18 b...	SUCCESS	b.	0
26	2007-06-09 01:53:36	reachright2	b..	790.46	b..	barney_toy,lego_toy	19 b...	1780,1631	b..	[reach barney_toy]	18 b...	SUCCESS	b.	0
27	2007-06-09 01:53:37	reachright	b..	790.46	b..	barney_toy,lego_toy	19 b...	1780,1631	b..	[reach barney_toy]	18 b...	SUCCESS	b.	0
17	2007-04-12 15:32:22	handshake	b..	0	b..	door_motion,face,...	27 b...	1228,126...	4 b..	[handshake door_motion]	23 b...	SUCCESS	b.	0
10	2007-03-09 14:08:26	reachright2	b..	0	b..	barney_toy	10 b...	1634	b..	[reach barney_toy]	18 b...	SUCCESS	b.	0
1	2007-02-16 14:12:13	reachright	b..	0	b..	lego_toy	8 b...	1682	b..	[reach lego_toy]	16 b...	SUCCESS	b.	0
2	2007-02-16 14:15:45	reachright	b..	0	b..	barney_toy	10 b...	1684	b..	[reach barney_toy]	18 b...	SUCCESS	b.	0
3	2007-02-16 14:44:22	reachright	b..	0	b..	barney_toy	10 b...	1682	b..	[reach barney_toy]	18 b...	SUCCESS	b.	0
4	2007-02-16 15:00:37	reachright	b..	0	b..	barney_toy	10 b...	1685	b..	[reach barney_toy]	18 b...	SUCCESS	b.	0
6	2007-02-18 15:35:17	reachright2	b..	0	b..	barney_toy	10 b...	1686	b..	[reach barney_toy]	18 b...	SUCCESS	b.	0
7	2007-02-18 14:40:01	reachleft	b..	0	b..	barney_toy	10 b...	1686	b..	[reach barney_toy]	18 b...	SUCCESS	b.	0
8	2007-02-18 14:46:35	reachright	b..	0	b..	barney_toy	10 b...	1684	b..	[reach barney_toy]	18 b...	SUCCESS	b.	0
9	2007-03-09 14:04:02	reachright	b..	0	b..	barney_toy	10 b...	1682	b..	[reach barney_toy]	18 b...	SUCCESS	b.	0
16	2007-04-12 15:00:02	handshake	b..	0	b..	door_motion,face,...	27 b...	1248,127...	4 b..	[handshake door_motion]	23 b...	SUCCESS	b.	0
15	2007-04-12 14:52:47	handshake	b..	0	b..	door_motion,face	16 b...	1267,1233	b..	[handshake door_motion]	23 b...	SUCCESS	b.	0
14	2007-04-03 14:02:07	track	b..	0	b..	barney_toy	10 b...	1387	b..	[track barney_toy]	18 b...	SUCCESS	b.	0
13	2007-03-14 15:17:16	reachleft	b..	0	b..	barney_toy	10 b...	1780	b..	[reach barney_toy]	18 b...	SUCCESS	b.	0
11	2007-03-09 14:14:11	reachright2	b..	0	b..	barney_toy	10 b...	1574	b..	[reach barney_toy]	18 b...	SUCCESS	b.	0
18	2007-05-09 18:44:22	reachright	b..	0	b..	red_bag	7 b...	1577	b..	[reach red_bag]	15 b...	SUCCESS	b.	0
31	2007-06-17 04:46:59	track	b..	0	b..	lego_toy	8 b...	1777	b..	[track lego_toy]	16 b...	SUCCESS	b.	0
30	2007-06-17 03:24:29	track	b..	0	b..	barney_toy	10 b...	1820	b..	[track barney_toy]	18 b...	SUCCESS	b.	0

Figure 34: Episodic Memory Contents for Experiments 1 and 2

CHAPTER V

EXPERIMENTAL RESULTS

Experiment 1

In this experiment ISAC Barney as a sphere located somewhere around his lower chest as seen in figure 35. Table 9 shows the actual SES values for Barney. This experiment has five trials – one trial for each stepping. In each trial the Barney toy was placed at the same SES coordinates and thus the same IRS Cartesian coordinates.

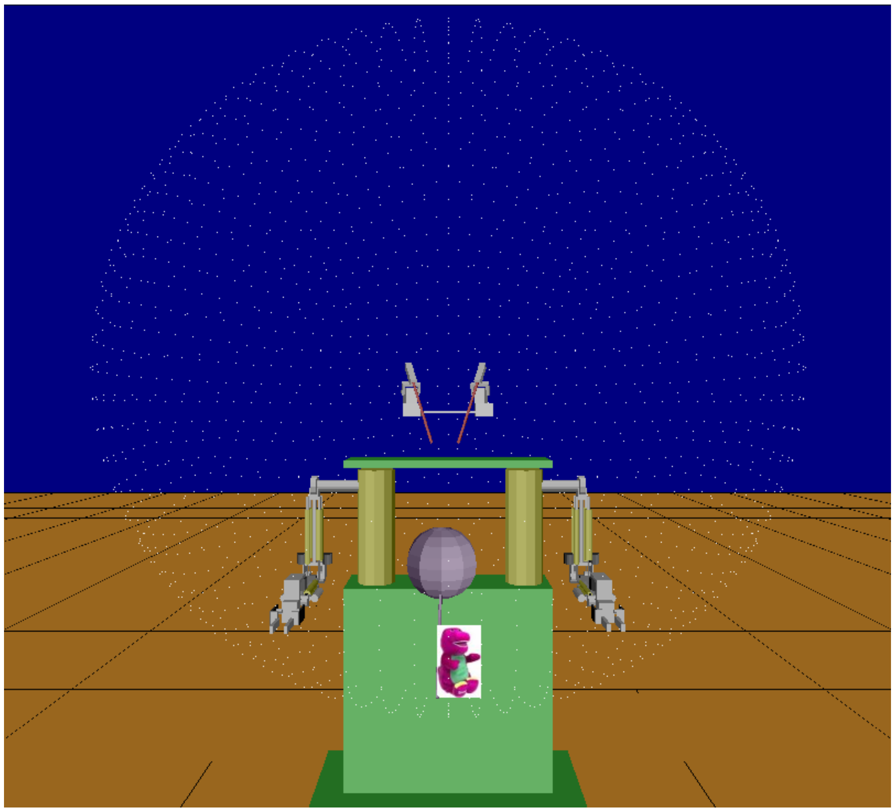


Figure 35: Exp 1 – ISAC Perceives Barney in IRS

Table 9: SES Information for Experiment 1

ID	Percept	Azimuth (deg)	Elevation (deg)	Distance (m)
1780	Barney_toy	-2.1611	139.93	0.82358

Trial 1

For each trial, ISAC was asked to reach to the Barney toy. Based on the assumptions for this experiment, ISAC will perform a regular right reach to the Barney toy. This response comes from knowledge initially stored in the First-Order Response Agent.

Table 10: Exp 1 – Experiment Timing (seconds) – Stepping of 1

Behavior	Arm Interpolation	Arm Motion	IRS Interpolation	IRS Collision
RightReach1	7.407	11.347	2.876	1.562
RightReach2	0	0	0	0
LeftReach1	0	0	0	0

As explained in chapter III, the Arm Agents and IRS have two phases of operation, the interpolation phase and the motion phase. During the interpolation phase, the arm trajectory is computed via Verbs and Adverbs (with a stepping of 1 for this trial). The second phase or motion phase is the phase where the arm or virtual arm moves. However, for IRS, the virtual arm moves and checks for collisions at the same time.

Table 10 shows the experiment timing for trial 1. This table shows the amount of time that each phase occurs for both the Arm Agents and IRS. *"RightReach1"* means regular reach with the Right Arm Agent. *"RightReach2"* means perform a reach around using the Right Arm Agent, and *"LeftReach1"* means perform a regular reach with the Left Arm Agent. A value of zero, in this table, means that the phase was not attempted.

When this task was given, FRA immediately placed the chunks "ReachRight1" and "Barney" into working memory. When the chunks appeared in WMS, the Activator Agent and Internal Rehearsal System processed these chunks simultaneously. The Right Arm Agent took 7.407 seconds to generate a trajectory towards the Barney percept, and IRS took 2.876 seconds to produce a similar trajectory towards this goal percept. IRS finished the collision detection phase within 1.562 seconds for a stepping of 1, and returned a prediction as shown in Table 11.

Table 11: Exp 1 – Reach Right IRS Predictions

Stepping	Joint Step	Joint Size	End Effector Travel (mm)	Collision Switches	Percept Collision
1	72	283	1006.18	1 0 0 0 0 0 0	Barney_toy
2	35	140	999.22	1 0 0 0 0 0 0	Barney_toy
3	23	92	965.45	1 0 0 0 0 0 0	Barney_toy
4	17	68	973.41	1 0 0 0 0 0 0	Barney_toy
5	14	54	988.45	1 0 0 0 0 0 0	Barney_toy

The first row shows the trial for the IRS predictions, and the stepping number for that trial. The second row of this table shows that the right end effector collision switch was tripped after the 72nd joint step in the interpolated right regular reach trajectory (with a complete joint size of 283). Because the collision percept was Barney, CEA considered this prediction a success. CEA did not change the working memory chunks, and thus AA was not suppressed. IRS's right regular reach behavior can be seen in figure 36. The Right Arm Agent executed the regular right reach behavior within 11.347 seconds. The other zeros in table 10 means that the Arm Agents and IRS did not perform the right reach around behavior or the left regular reach behavior.

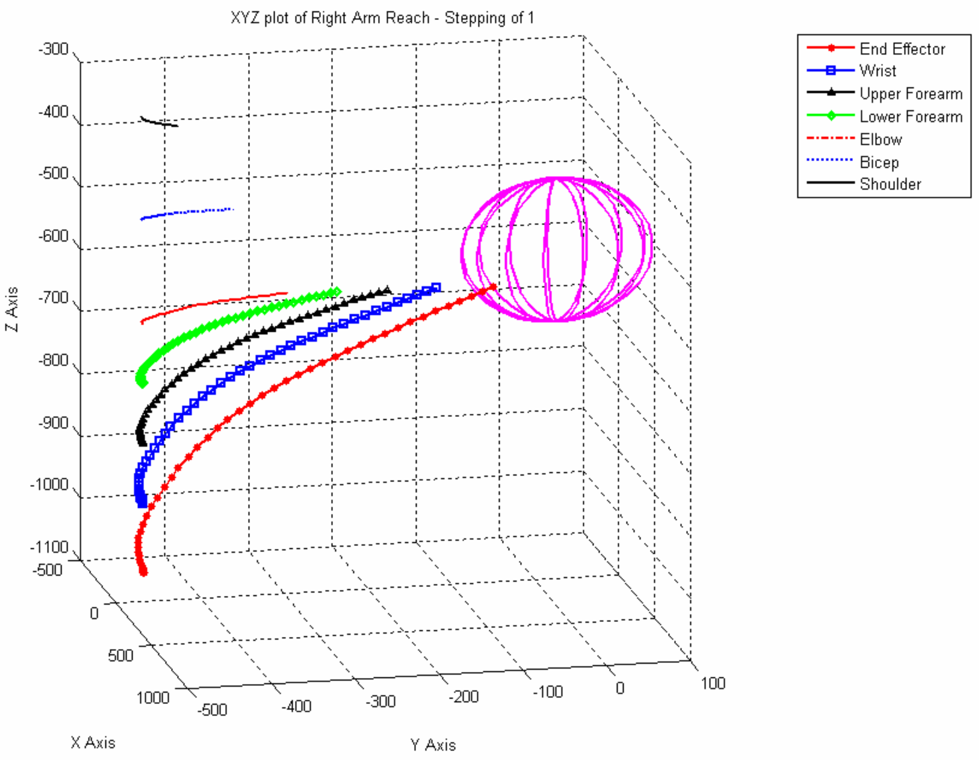


Figure 36: Exp 1 – Right Reach Behavior – Stepping of 1

Trial 2

Table 12: Exp 1 – Experiment Timing (seconds) – Stepping of 2

Behavior	Arm Interpolation	Arm Motion	IRS Interpolation	IRS Collision
RightReach 1	7.025	11.307	1.437	0.766
RightReach 2	0	0	0	0
LeftReach1	0	0	0	0

In trial 2, ISAC was given the same task to reach to Barney doll with no obstacle present. There are some differences between trials 1 and 2. Because of the increase in stepping (from 1 to 2), the IRS interpolation and collision times were almost twice as fast. Also, the IRS trajectory had a smaller joint size as seen in table 11; this caused sparseness in the collision point trajectories (figure 37). One unusual thing that happened during the trial was the slight change in arm interpolation time. The Right and Left Arm Agent computer was under different loads during the experimentation. This caused the variations in arm interpolation time. This trial was a success because IRS predicted that the regular reach right behavior will reach to Barney. IRS predicted a collision with the right end effector collision point in the 35th joint step.

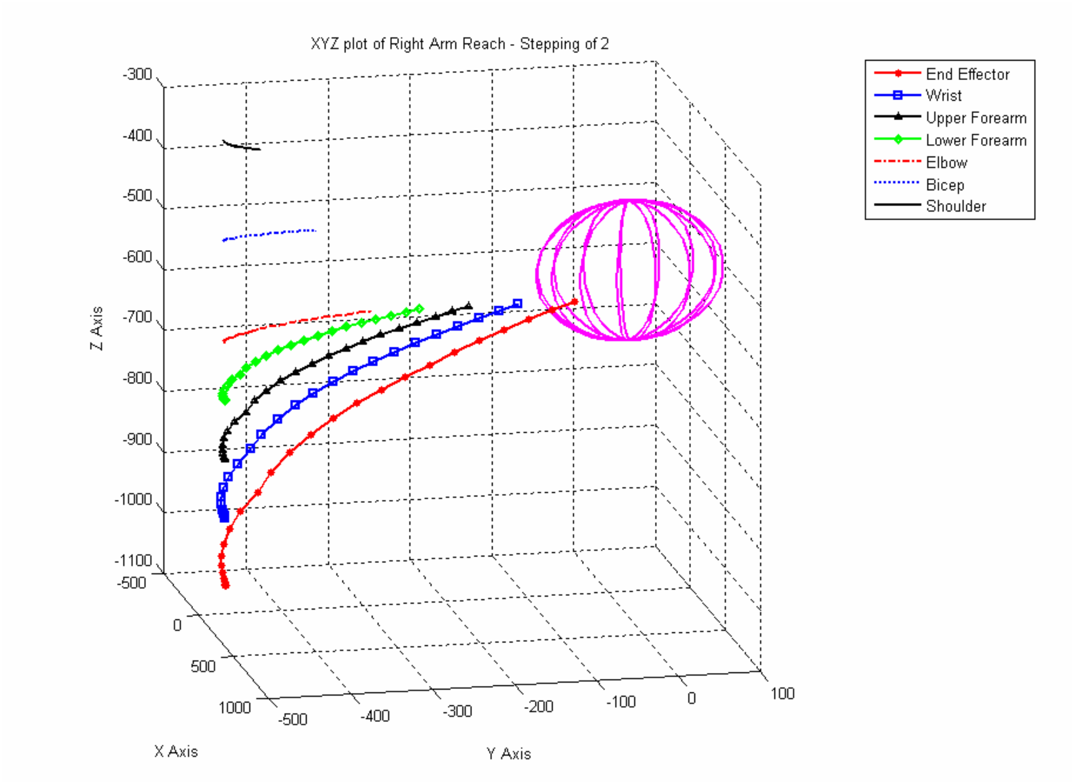


Figure 37: Exp 1 – Right Reach Behavior – Stepping of 2

Trial 3

Table 13: Exp 1 – Experiment Timing (seconds) – Stepping of 3

Behavior	Arm Interpolation	Arm Motion	IRS Interpolation	IRS Collision
RightReach1	7.606	11.437	0.938	0.609
RightReach2	0	0	0	0
LeftReach1	0	0	0	0

Trial 3 was also successful. Table 13 shows that, once again, the IRS interpolation and collision timing further speeded up with a higher stepping. This table also shows that ISAC reached to the Barney using the right regular reach. Figure 38 shows the IRS arm trajectory; the right end effector collision point entered the Barney sphere on the 23rd joint step as shown in table 11.

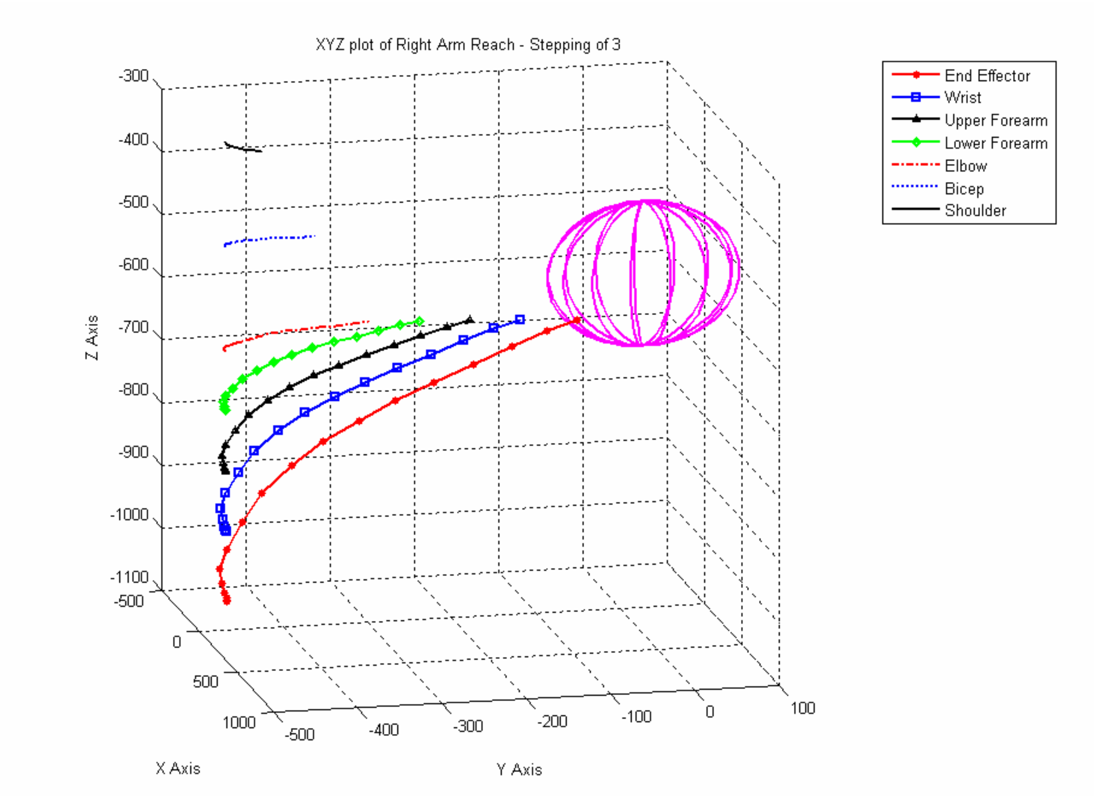


Figure 38: Exp 1 – Right Reach Behavior – Stepping of 3

Trial 4

Table 14: Exp 1 – Experiment Timing (seconds) – Stepping of 4

Behavior	Arm Interpolation	Arm Motion	IRS Interpolation	IRS Collision
RightReach1	7.546	11.627	0.688	0.469
RightReach2	0	0	0	0
LeftReach1	0	0	0	0

Trial 4 was successful like the previous three trials. IRS speeded up approximately four times in both the interpolation and collision phases. Also, ISAC successfully reached to Barney using its right arm. The IRS arm trajectory is shown in figure 39.

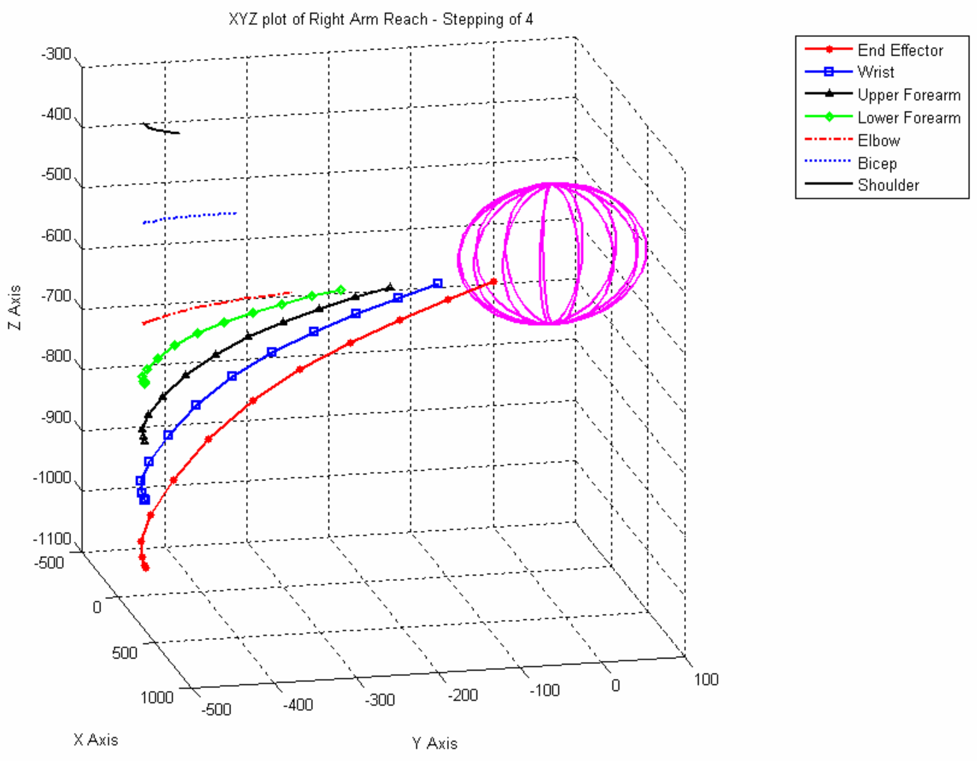


Figure 39: Exp 1 – Right Reach Behavior – Stepping of 4

Trial 5

Table 15: Exp 1 – Experiment Timing (seconds) – Stepping of 5

Behavior	Arm Interpolation	Arm Motion	IRS Interpolation	IRS Collision
RightReach1	7.606	11.457	0.531	0.219
RightReach2	0	0	0	0
LeftReach1	0	0	0	0

Trial 5 produced the fastest times for IRS. IRS had an interpolation time of 0.531 seconds and a collision time of 0.219 seconds. Also, the trajectory produced by the Verbs and Adverbs algorithm was the smallest with a joint size of 54 as shown in table 11. This trial seems to be the best, simply because IRS was able to produce a prediction in the smallest time period. However, because every trial successfully predicted a collision with the Barney toy, the question arises, "what will happen with steppings above 5"? This question is further discussed in the discussion section.

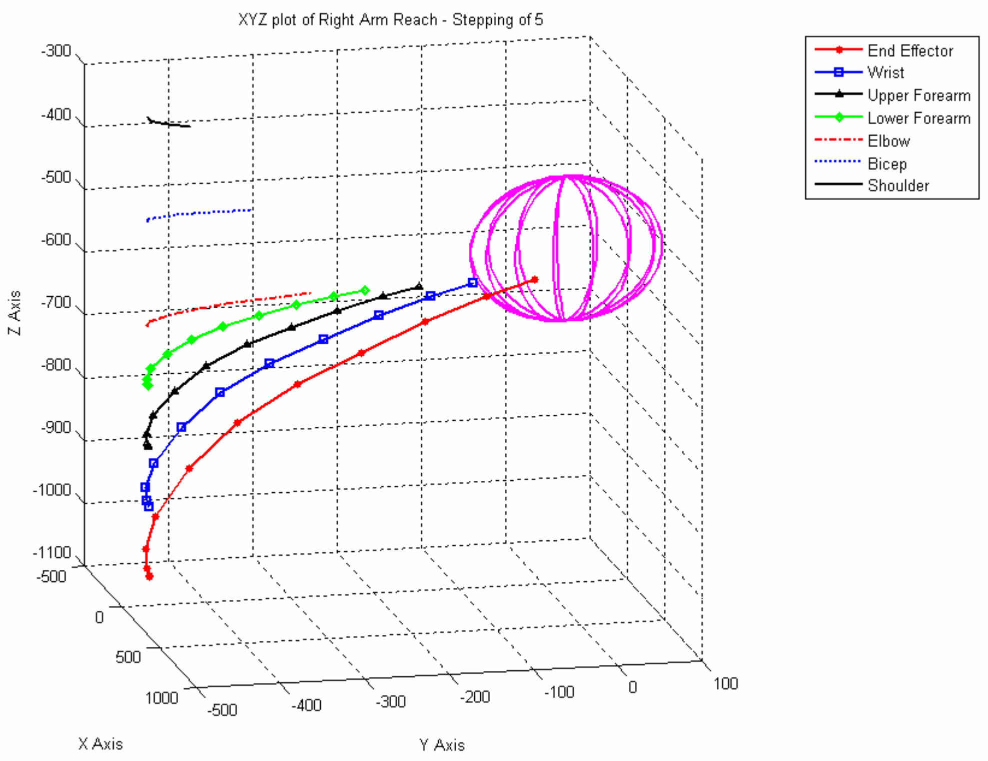


Figure 40: Exp 1 – Right Reach Behavior – Stepping of 5

Discussion of Results

For experiment 1, all five trials were successful in terms of prediction. CEA received a positive prediction from IRS and did not interfere with the FRA's working memory chunks. In each case, ISAC reached to the Barney toy by using the regular reach right behavior. The other two behaviors, right reach around and regular left reach were not necessary to satisfy the goal. The process used to complete the task, "Reach to Barney" for each trial can be seen in figure 41.

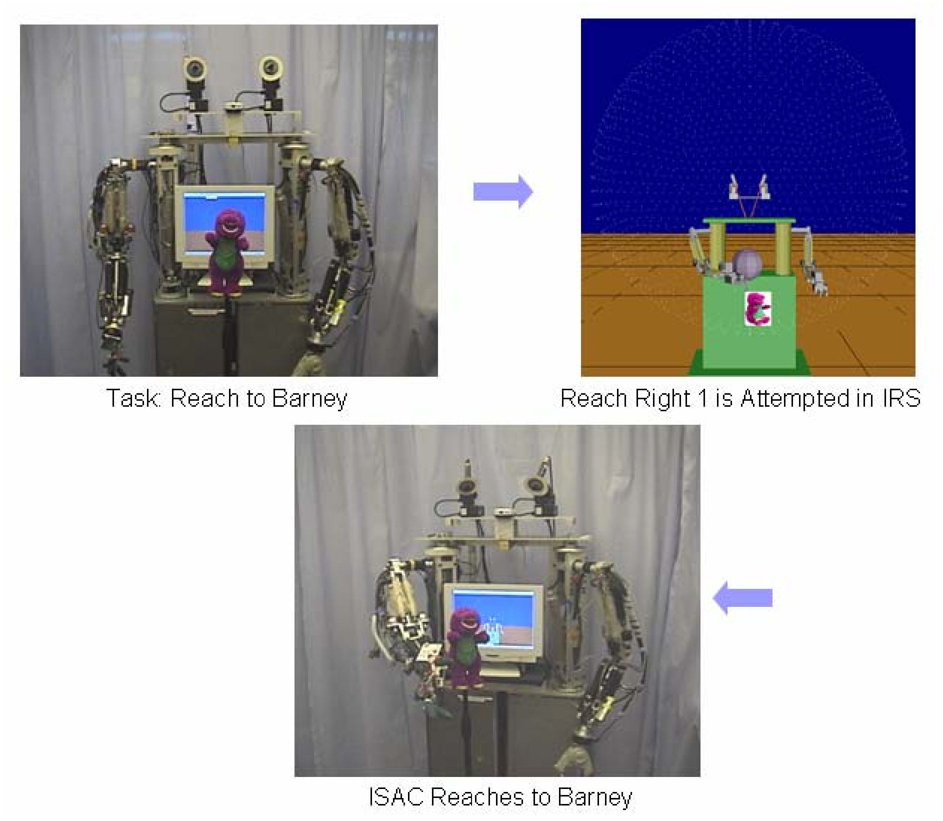


Figure 41: Exp 1 – ISAC Reaches to Barney Using IRS (Best Case Scenario)

Because every trial was successful, it raises questions about higher stepping levels. Why not try steppings of 6, 7, 8... etc. Well, IRS has one problem that was uncovered in this experiment. The second behavior, reach right around, does not work past a stepping of 5. This is because stepping takes information away from the raw example motion files used for the

Verb/Adverb interpolation. The right reach around is a complicated motion (shown in the next experiment). It has more motion exemplars bounded by keytimes. Stepping past 5 makes the raw motion files lose enough resolution to where too many keytimes exist relative to the motion exemplars. According to [Spratley, 2006], if a motion has too many or not enough keytimes, the motion will not be accepted by the Verb/Adverb algorithm. Even though this problem exists, steppings 10, 15, 20, 25, and 30 for the regular right reach were tried offline. The information in figures 42 and 43 was revealed.

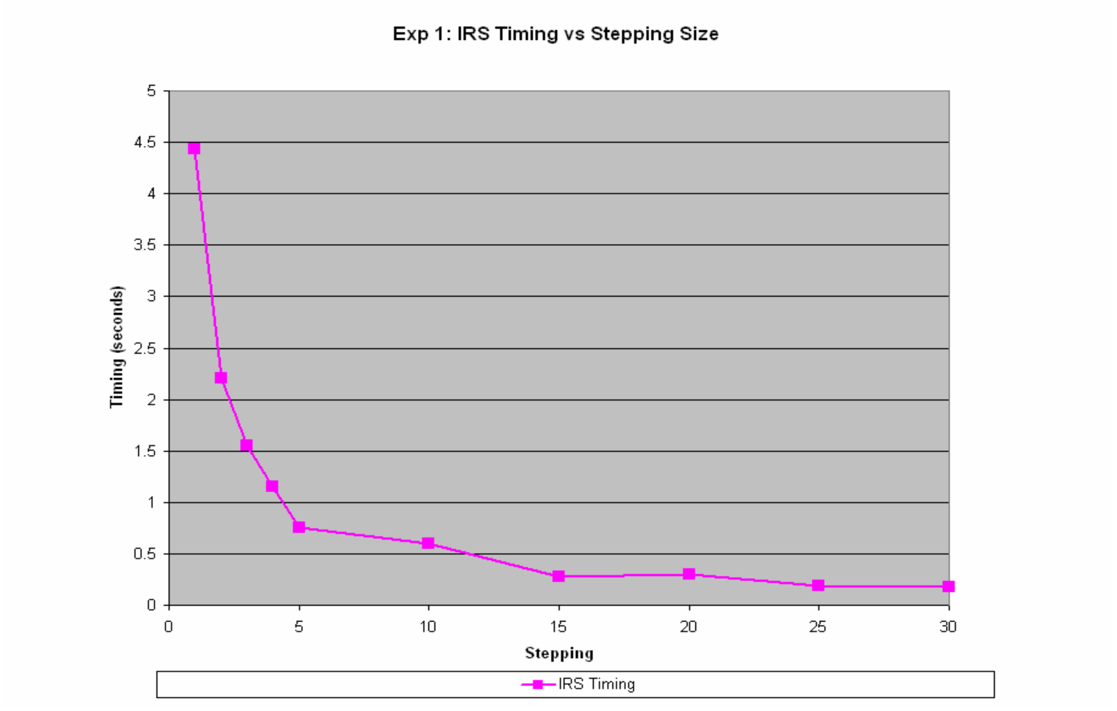


Figure 42: Exp 1 – IRS Timing vs Stepping Size

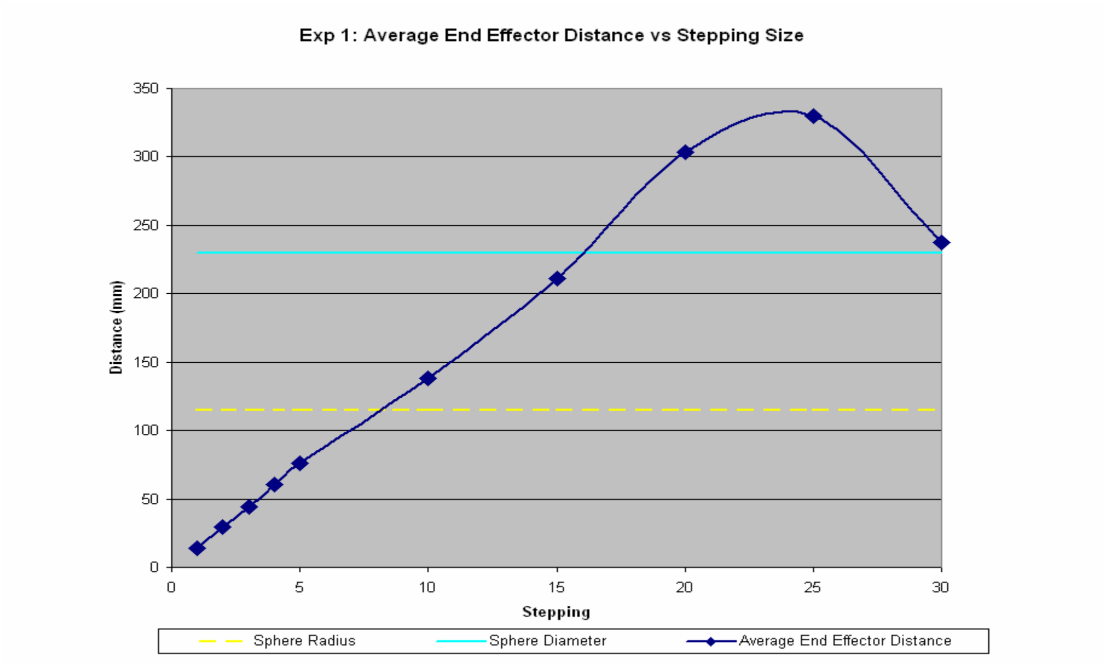


Figure 43: Exp 1 – Average End Effector Distance vs Stepping Size

Figure 42 shows a graph of IRS timing vs stepping size for this experiment. The line with squares shows the total IRS timing (interpolation time plus collision time). Figure 43 shows a graph of average end effector distance vs stepping size for experiment 2. The dashed line in figure 43 is the radius of the collision spheres, and the solid line is the diameter of the collision spheres used in both experiments 1 and 2. The average end effector stepping distance line (diamond line) shows that there is a linear relationship between stepping and average end effector stepping distance (y=15x). This is to be expected. However, around a stepping of 15, the stepping distance becomes greater than the diameter of the collision spheres used. This is unacceptable because an entire collision sphere could be skipped during the internal rehearsal. In general though, it is best to keep the stepping distance as low as possible because the Verb and Adverb motion will not always hit a collision sphere squarely. Even though the steppings 1 through 15 are acceptable, the steppings past five are not desireable simply because the timing goes down slightly for a large traveling distance as seen in figure 42.

Experiment 2

In this experiment, two objects were presented to ISAC, the Barney doll and the Lego toy. IRS perceives the percept locations from the SES as seen in figure 44. The objects are located at the SES coordinates as shown in table 16. As with Experiment 1, this experiment has five trials trying all five steppings. Also, the objects are stationary, so the SES coordinates are the same for each trial.

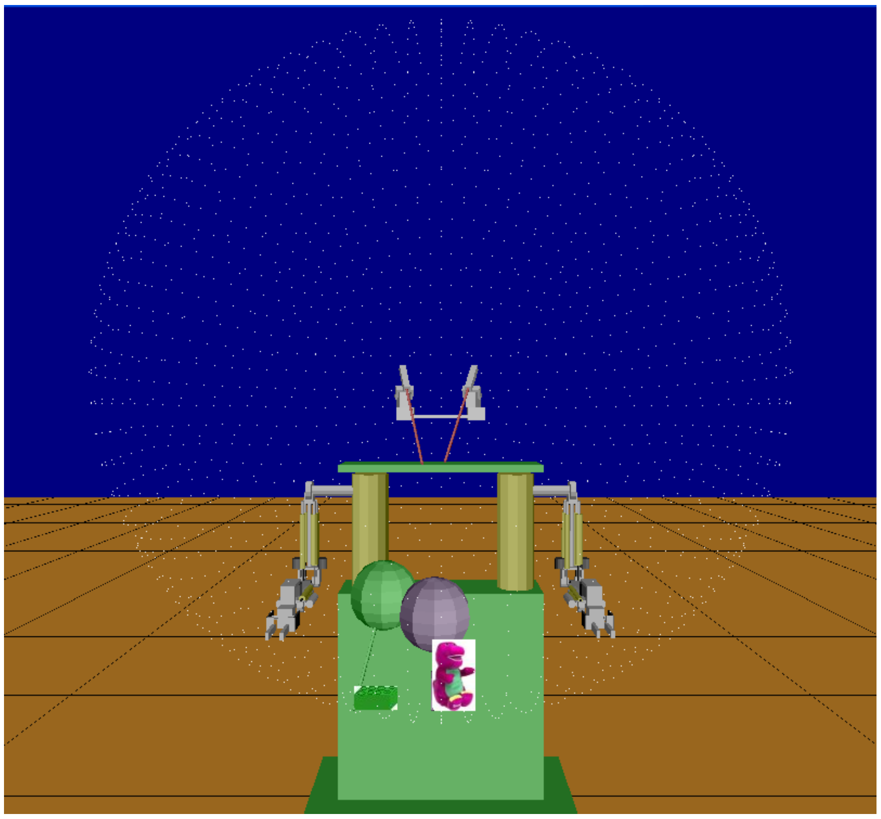


Figure 44: Exp 2 – ISAC Perceives Barney and Lego toy in IRS

Table 16: SES Information for Experiment 2

ID	Percept	Azimuth (deg)	Elevation (deg)	Distance (m)
1780	Barney_toy	-2.0096	143.13	0.95021
1818	Lego_toy	-23.963	147.99	0.8723

Trial 1

Table 17: Exp 2 – Experiment Timing (seconds) – Stepping of 1

Behavior	Arm Interpolation	Arm Motion	IRS Interpolation	IRS Collision
Right Reach 1	7.542	0	2.765	3.282
Right Reach 2	11.687	**2.655**	4.313	9.875
Left Reach 1	7.792	12.599	2.735	4.493

Table 17 shows the timing information for the behaviors attempted by both IRS and Arm Agents. It shows that the Right Arm Agent took 7.542 seconds to interpolate a trajectory to perform a regular right reach to Barney. At the same time, IRS interpolated this reach (with a stepping of 1) in 2.765 seconds. After the interpolations were calculated, IRS performed the reach and found a collision within 3.282 seconds, and a prediction was sent to CEA. The predictions for all three behaviors are in table 18. The time of 2.765 seconds plus 3.282 seconds is less than the Right Arm Agent's interpolation time, and as seen in the first row of table 17, IRS sent back a wrist collision with the Lego toy at the sixty-fourth joint step out of 273 of the interpolated motion. CEA saw this as not accomplishing ISAC's goal and suppressed AA.

As discussed in chapter IV, CEA chose another behavior from its behavior list, right reach around in this case. This behavior was chosen because this behavior had the second most experience in the Episodic Memory. Table 18 shows that it returns similar results, but during the internal rehearsal, the upper forearm collision switch was triggered. Also, the end effector distance was much longer meaning that this behavior traveled much farther to the Lego percept than the regular reach. Unfortunately, during the right reach around behavior, a problem occured. The problem was that IRS took too long, and as seen in table 17 in bold, the arm moved for 2.655 seconds. This means that CEA was unable to stop the Right Arm Agent before it moved the arm. As explained in the IRS Stepping section of chapter III, IRS must finish both phases before the Arm Agent is able to finish its first phase. Otherwise, the arm will begin to move before CEA is able to replace the chunks in working memory causing a worst case scenario.

After 2.655 seconds of right arm movement, CEA suppressed the Activator Agent and stoped the arm. CEA then pulled its last behavior out of the list. This behavior succeeded as seen in table 18. IRS determined that the end effector reached the Barney sphere. Therefore, CEA did not suppress the Activator Agent, and the end result was that ISAC reached to Barney with the left arm.

Figures 45, 46 and 47 show the plotted movement of ISAC's arms during the right regular reach, right reach around, and left regular reach respectively. Notice how close the markings are together from the elbow to the end effector. This means that the interpolated motion trajectory during the collision detection is very fine. Figure 45 clearly shows that the right wrist collision point, shown as a line with squares, enters the Lego sphere. Figure 46 shows that the right upper forearm collision point enters the Lego sphere; this line contains triangles. Finally, figure 47 displays that the left end effector point enters the Barney sphere. This information from the figures is consistent with the tripped collision switches in table 18.

Table 18: Exp 2 – Trial 1 – IRS Prediction

Behavior	Joint Step	Joint Size	End Effector Travel (mm)	Collision Switches	Percept Collision
Right Reach 1	64	273	776.88	0 1 0 0 0 0 0	Lego_toy
Right Reach 2	157	425	1420.37	0 0 1 0 0 0 0	Lego_toy
Left Reach 1	76	269	922.87	1 0 0 0 0 0 0	Barney_toy

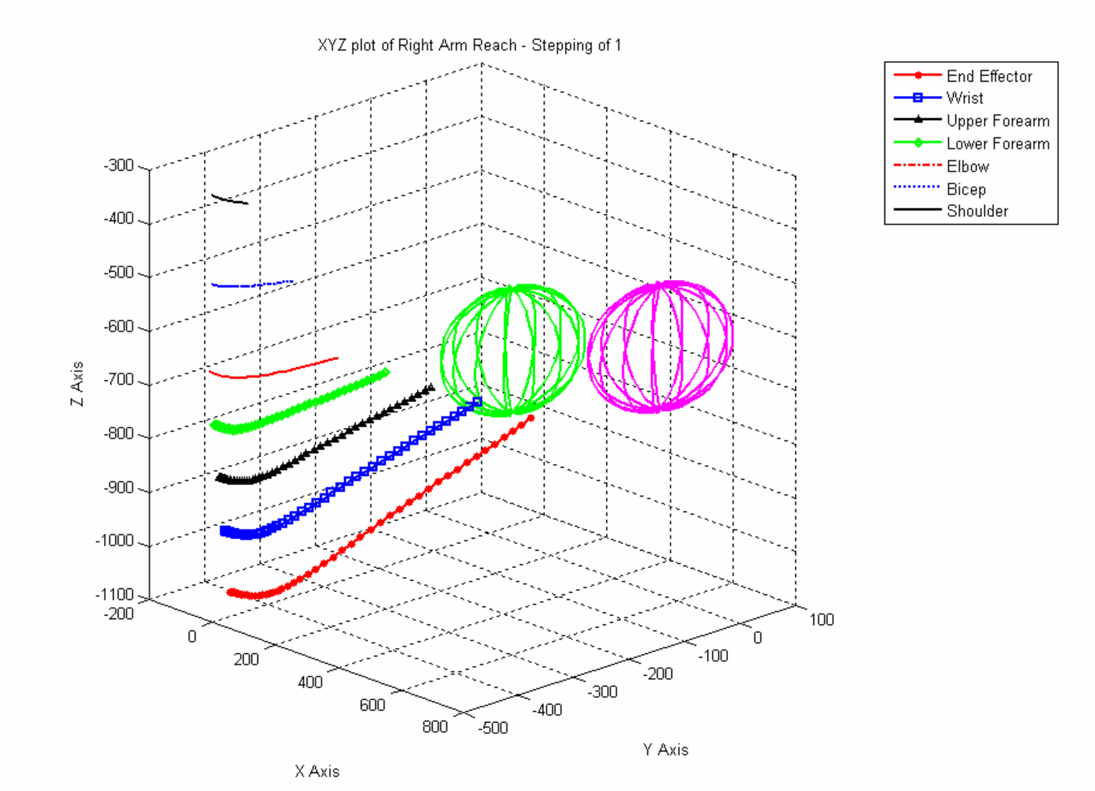


Figure 45: Exp 2 – Right Reach Behavior – Stepping of 1

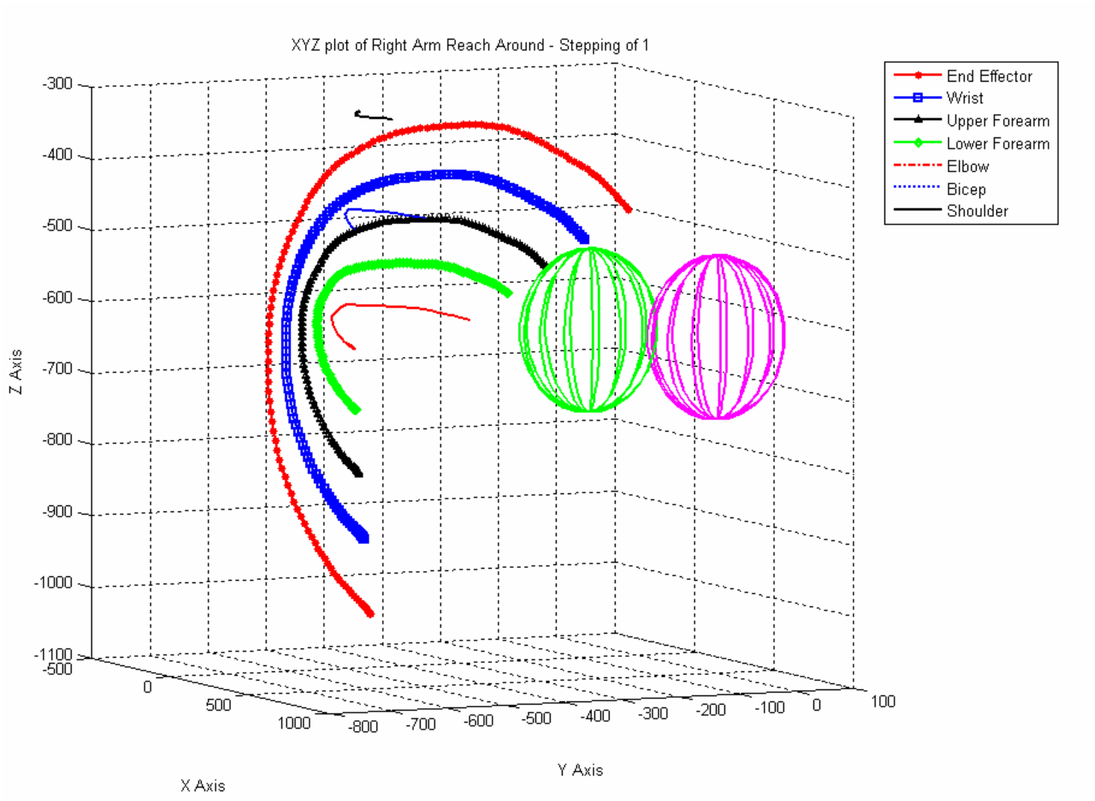


Figure 46: Exp 2 – Right Reach Around Behavior – Stepping of 1

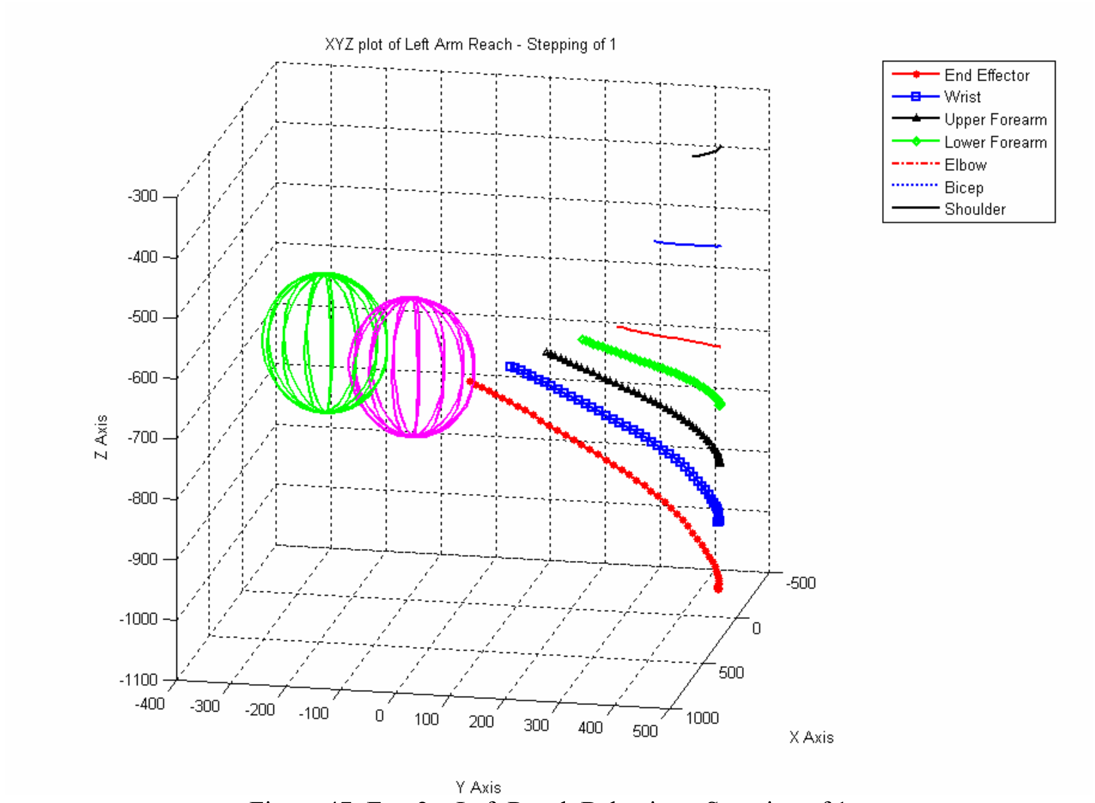


Figure 47: Exp 2 – Left Reach Behavior – Stepping of 1

<u>Trial 2</u>

Table 19: Exp 2 – Experiment Timing (seconds) – Stepping of 2

Behavior	Arm Interpolation	Arm Motion	IRS Interpolation	IRS Collision
Right Reach 1	7.752	0	1.343	1.829
Right Reach 2	13.350	0	2.110	4.515
Left Reach 1	6.891	11.397	1.312	1.531

Table 20: Exp 2 – Trial 2 – IRS Prediction

Behavior	Joint Step	Joint Size	End Effector Travel (mm)	Collision Switches	Percept Collision
Right Reach 1	31	134	753.71	0 1 0 0 0 0 0	Lego_toy
Right Reach 2	80	211	1432.92	0 0 1 0 0 0 0	Lego_toy
Left Reach 1	37	132	908.33	1 0 0 0 0 0 0	Barney_toy

For trial 2, ISAC was given the same task to reach to the Barney toy. Table 19 shows that this trial was successful. IRS was able to give a prediction back to CEA within a timely manner. This can be seen in the arm motion column. The arm did not move until it is given the command to reach to the Barney using the left arm. CEA was able to suppress the Activator Agent before the arm could move. As with trial 1, CEA received a Lego collision prediction for the right reach and the right reach around. Also, the right virtual arm striked two different areas of the arm, the wrist for the regular reach and the upper forearm for the right reach around (table 20).

Table 20 also shows the effect of the stepping. IRS produced a trajectory with a much smaller joint size. This allowed IRS to check for collisions and interpolate much more quickly. The plots for IRS are shown in figures 48, 49, and 50 for the regular right reach, right reach around, and the regular left reach respectively. In this trial, ISAC reached to Barney with its left arm in 11.397 seconds.

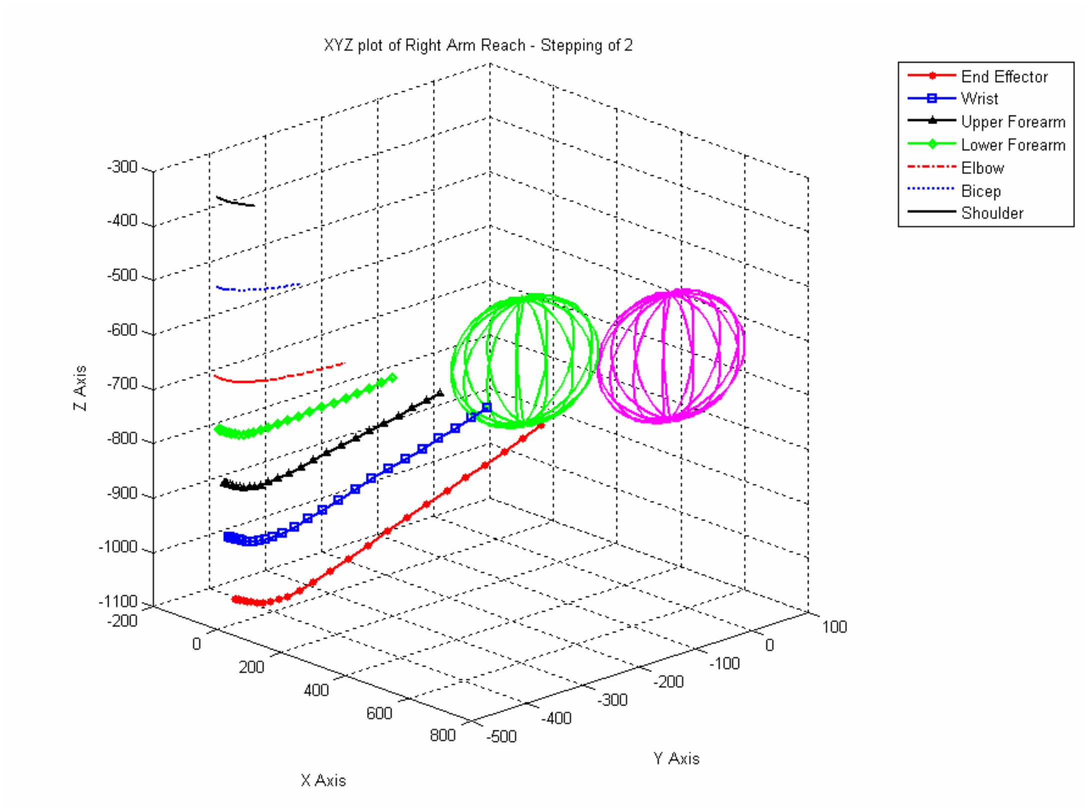


Figure 48: Exp 2 – Right Reach Behavior – Stepping of 2

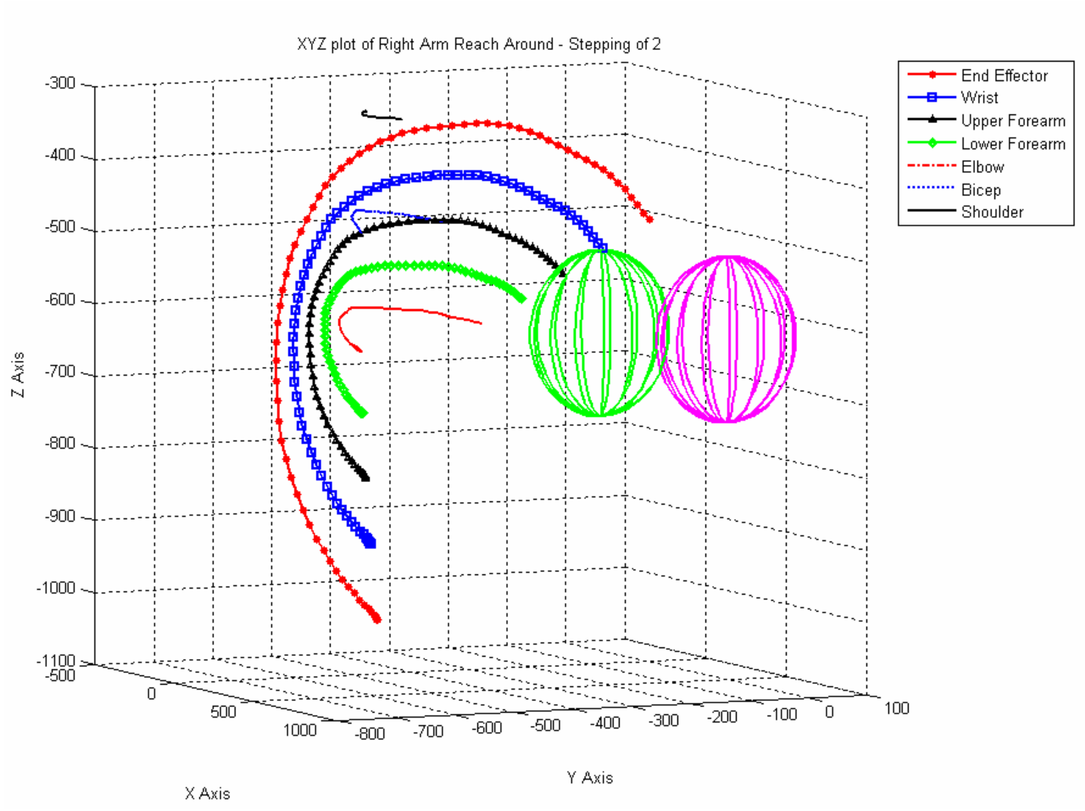


Figure 49: Exp 2 – Right Reach Around Behavior – Stepping of 2

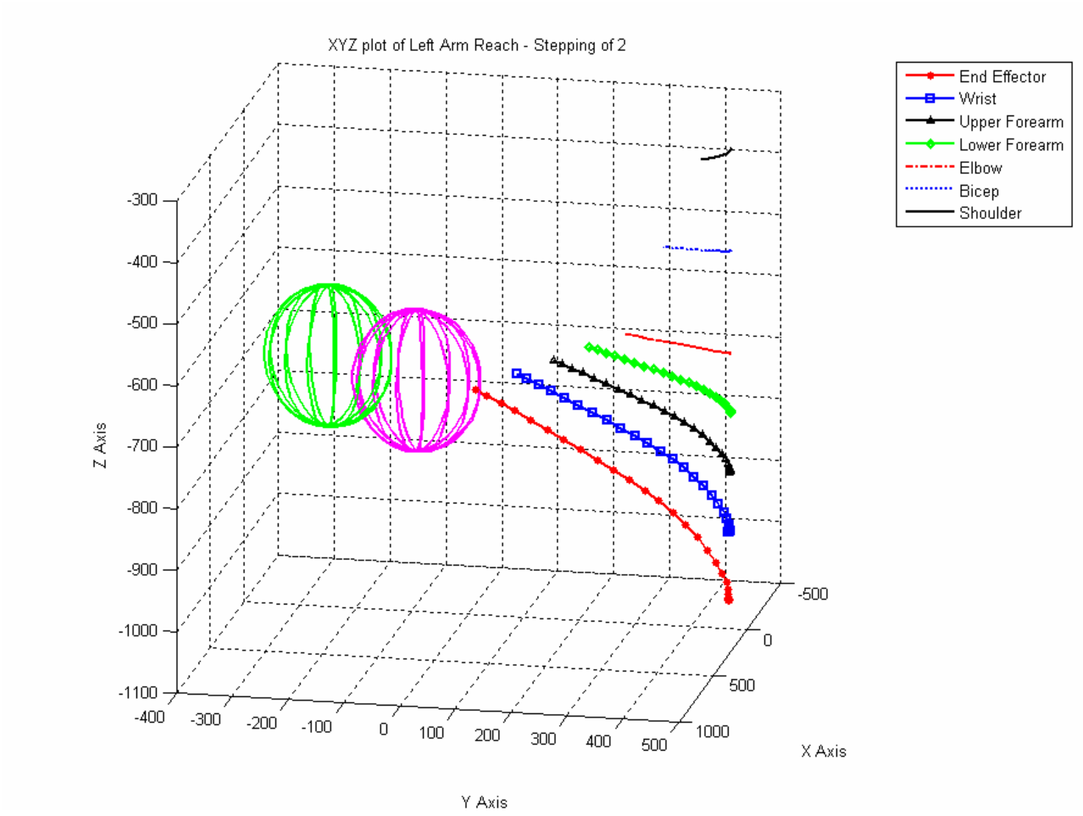


Figure 50: Exp 2 – Left Reach Behavior – Stepping of 2

Trial 3

Table 21: Exp 2 – Experiment Timing (seconds) – Stepping of 3

Behavior	Arm Interpolation	Arm Motion	IRS Interpolation	IRS Collision
Right Reach 1	7.031	0	0.921	1.219
Right Reach 2	12.348	0	1.391	2.672
Left Reach 1	7.431	12.679	0.860	1.062

Table 22: Exp 2 – Trial 3 – IRS Prediction

Behavior	Joint Step	Joint Size	End Effector Travel (mm)	Collision Switches	Percept Collision
Right Reach 1	21	89	761.10	0 1 0 0 0 0 0	Lego_toy
Right Reach 2	52	139	1425.78	0 0 1 0 0 0 0	Lego_toy
Left Reach 1	23	87	906.02	1 0 0 0 0 0 0	Barney_toy

In this trial, ISAC was asked to reach to the Barney toy with the Lego obstacle present. As with trial 2, ISAC succeeded. The arm did not move for the right reach “RightReach1” or reach around “RightReach2”, because IRS predicted a collision with the the Lego_toy in both cases. Also, within IRS, the right virtual arm collided with the Lego percept with the wrist (for the regular reach), and the upper forearm (for the reach around), but the left virtual arm reached to the Barney and tripped the end effector collision point. As with both trials 1 and 2, ISAC reached to the Barney using its actual left arm in 12.679 seconds. Figures 51, 52, and 53 show the trajectories generated by IRS for this trial.

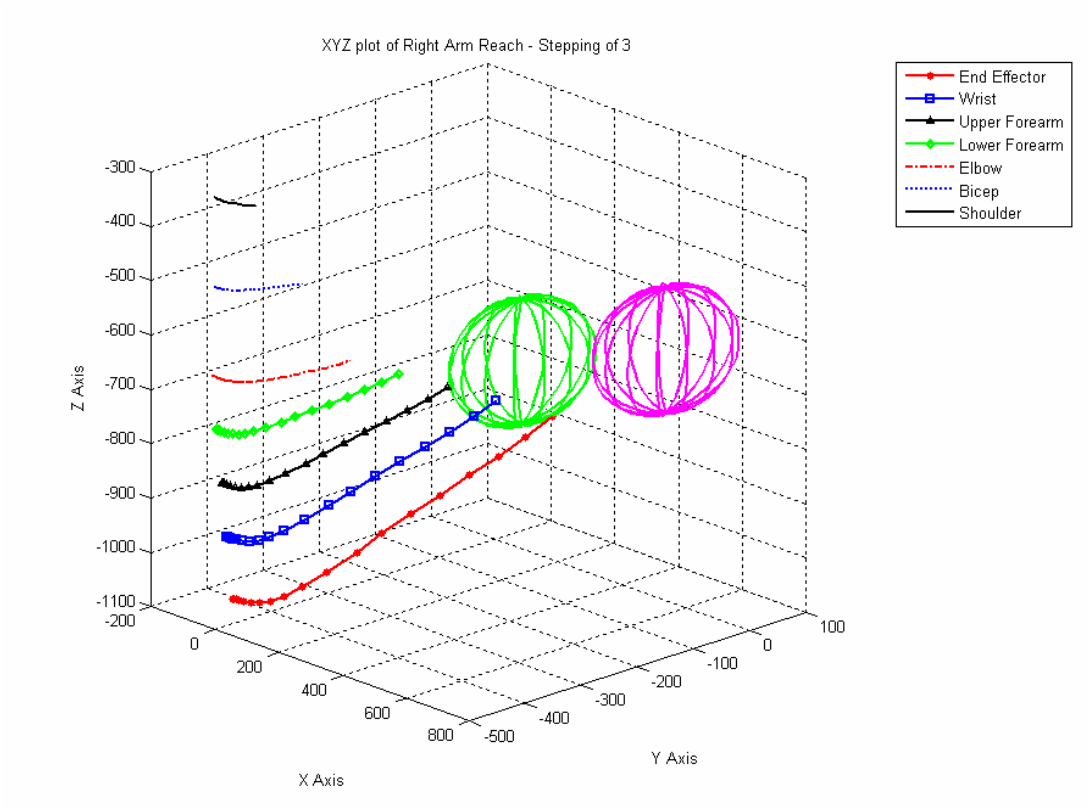


Figure 51: Exp 2 – Right Reach Behavior – Stepping of 3

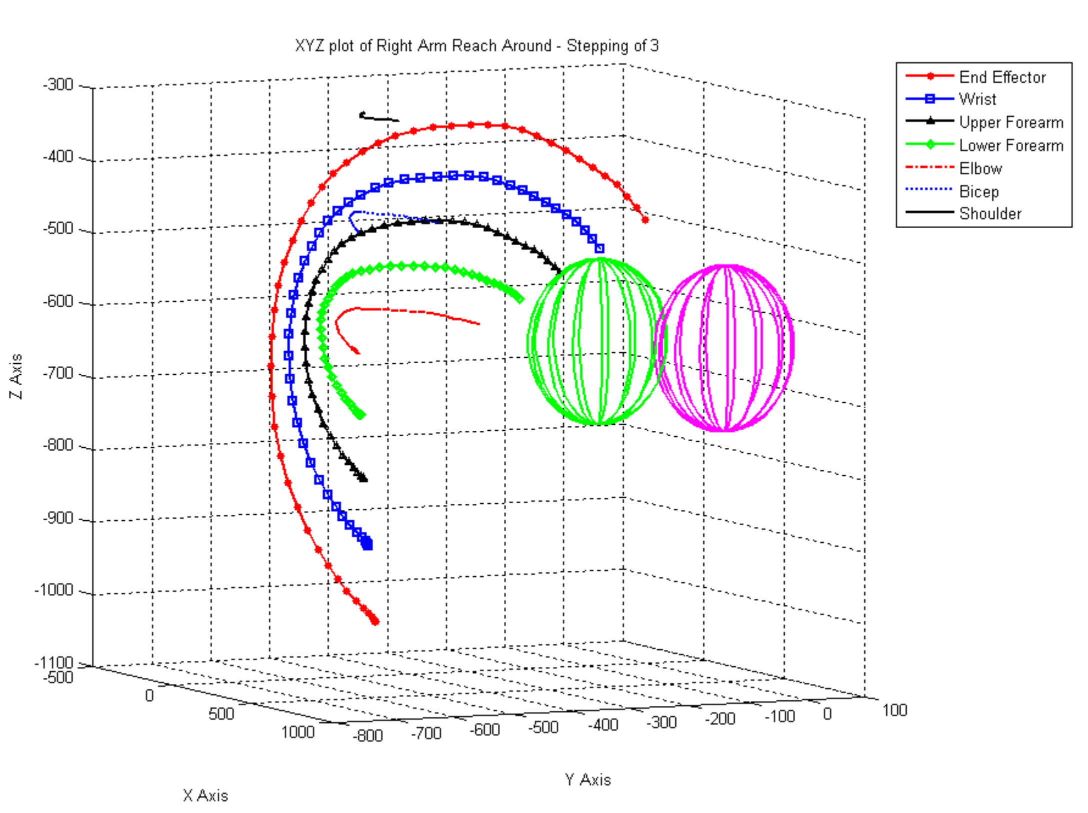


Figure 52: Exp 2 – Right Reach Around Behavior – Stepping of 3

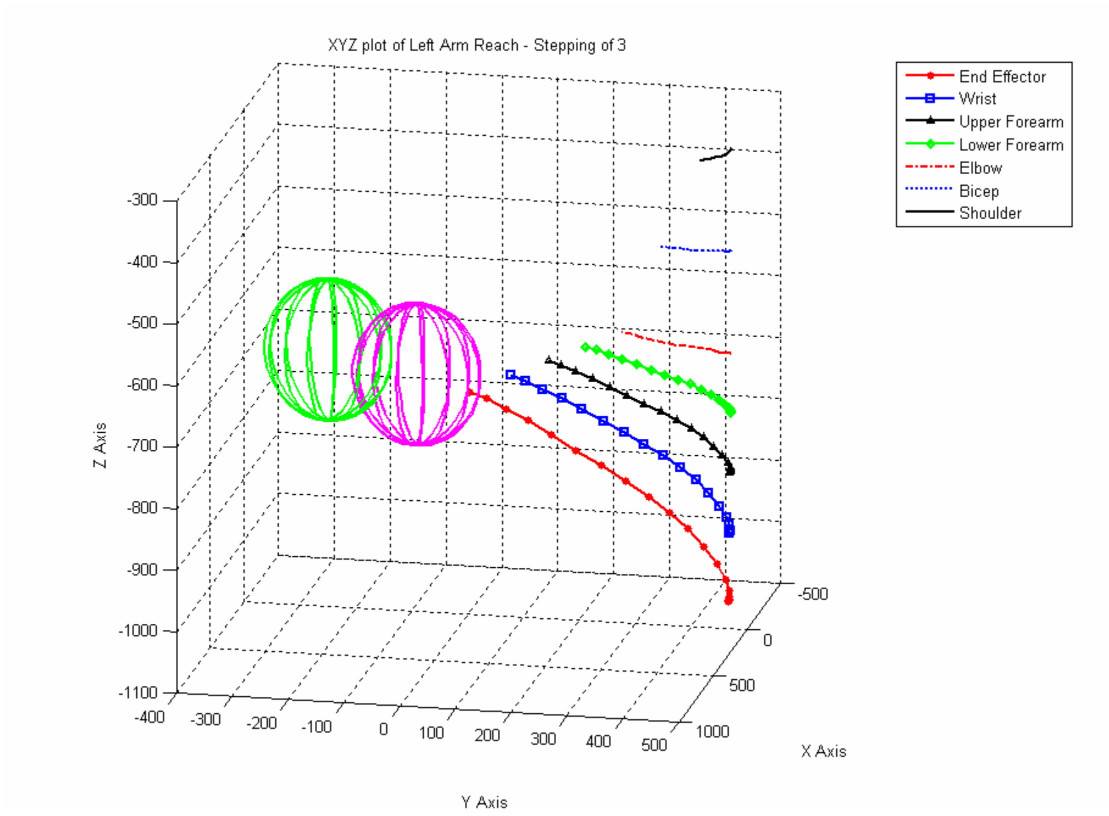


Figure 53: Exp 2 – Left Reach Behavior – Stepping of 3

Trial 4

Table 23: Exp 2 – Experiment Timing (seconds) – Stepping of 4

Behavior	Arm Interpolation	Arm Motion	IRS Interpolation	IRS Collision
Right Reach 1	7.150	0	0.656	0.844
Right Reach 2	0	0	1.031	2.297
Left Reach 1	8.652	11.568	0.640	0.985

Table 24: Exp 2 – Trial 4 – IRS Prediction

Behavior	Joint Step	Joint Size	End Effector Travel (mm)	Collision Switches	Percept Collision
Right Reach 1	15	65	749.02	0 1 0 0 0 0 0	Lego_toy
Right Reach 2	40	104	1452.28	0 1 1 0 0 0 0	Lego_toy
Left Reach 1	18	65	931.43	1 0 0 0 0 0 0	Barney_toy

Trial 4 produced somewhat different results than the previous three trials. Once again, ISAC was asked to reach to Barney with an obstacle present. In this trial, IRS had become so fast that the Right Arm Agent was simply not able to keep up. In table 23, there is a zero in the Arm Interpolation column. This means that IRS was so fast that CEA was able to suppress the Activator Agent twice before the Right Arm Agent could complete the first arm interpolation. During the interpolation phase, the Right Arm Agent did not check for new orders from the Activator Agent. Only after this phase was finished did it recheck with AA. As see in table 23, IRS completed calculations for both right reaches within 4.828 seconds. There is an extra time of about a second where CEA is checking EM, replacing chunks in WMS, and communicating with AA. However, this time is below the 7.150 seconds of arm interpolation. Thus, the Right Arm Agent did not know that the second command was issued by AA. The Left Arm Agent did acknowledge the “ReachLeft1” command though because it was the last command issued by the AA, and thus timing was not an issue.

Also, another difference was the fact that the joint sizes became so small that the virtual arm moved far ahead each joint step. This caused both the wrist and upper forearm collision switches to trip during internal rehearsal (table 24). Even though the stepping of 4 caused these two problems, ISAC still performed the correct action. It reached to Barney using the regular left reach behavior via the Left Arm Agent. Figures 54 through 56 show the IRS trajectories for each behavior in this trial.

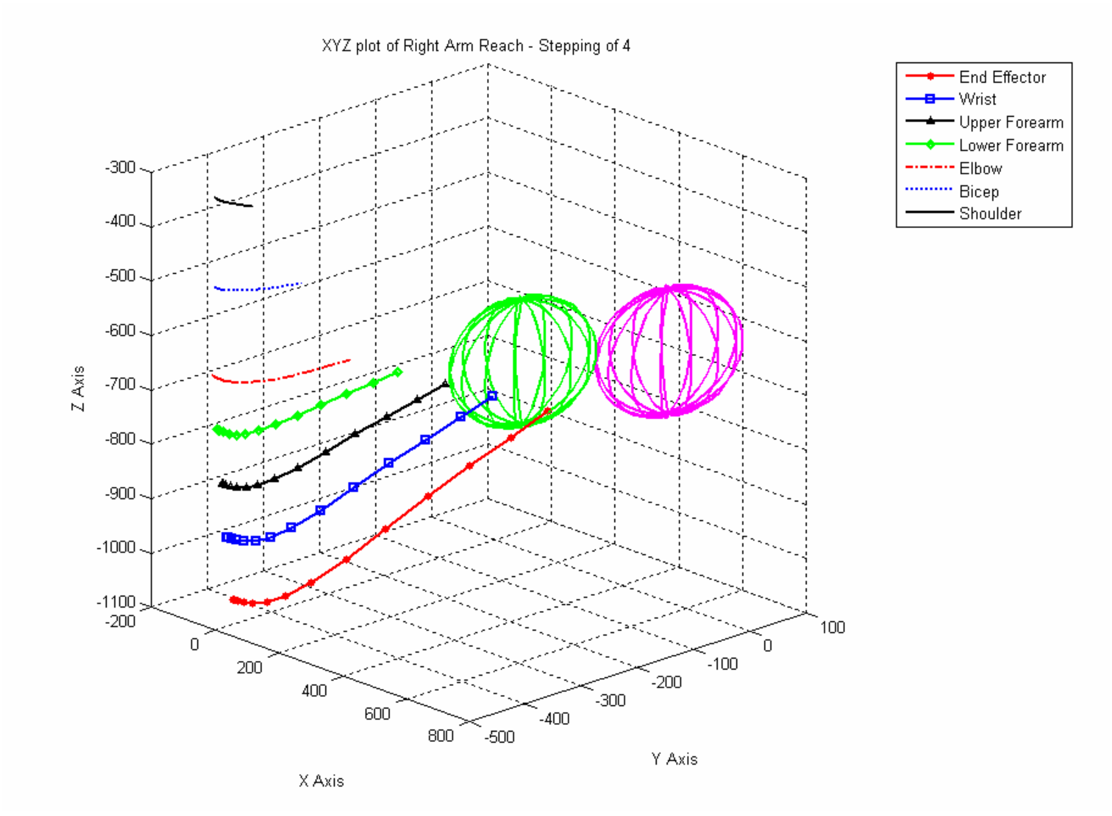


Figure 54: Exp 2 – Right Reach Behavior – Stepping of 4

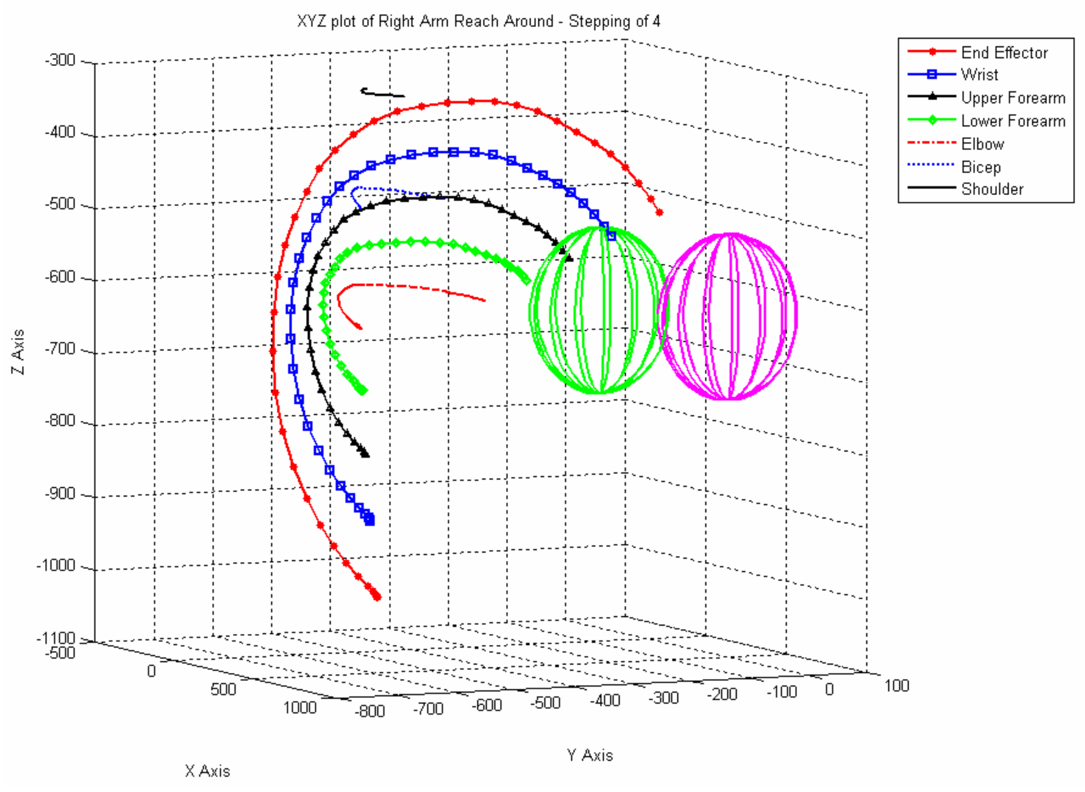


Figure 55: Exp 2 – Right Reach Around Behavior – Stepping of 4

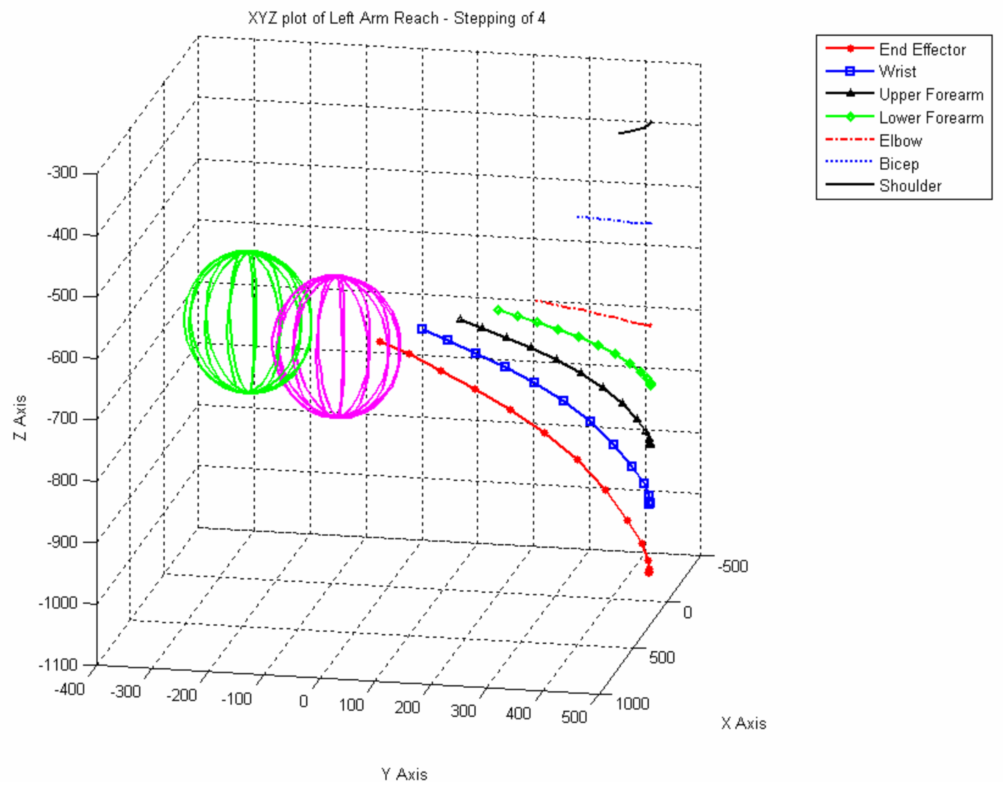


Figure 56: Exp 2 – Left Reach Behavior – Stepping of 4

Trial 5

Trial 5 produced very similar results to trial 4. Once again, IRS produced predictions so quickly that the Right Arm Agent was not fast enough to keep up in terms of interpolation time. The Right Arm Agent did not interpolate the reach right around motion “ReachRight2” for the exact same reason as Trial 4. Also, the virtual arm trajectory moved so quickly that both the wrist and upper forearm collision switches tripped. As with the previous four trials, CEA was successful in determining the correct behavior for the given task. ISAC reached to Barney using its left arm.

Figures 57-59 show the collision point trajectories for the regular right reach, the right reach around, and the regular left reach respectively.

Table 25: Exp 2 – Experiment Timing (seconds) – Stepping of 5

	Arm Interpolation	Arm Motion	IRS Interpolation	IRS Collision
Right Reach 1	7.150	0	0.532	0.765
Right Reach 2	0	0	0.844	1.953
Left Reach 1	5.659	11.036	0.640	0.985

Table 26: Exp 2 – Trial 5 – IRS Prediction

Behavior	Joint Step	Joint Size	End Effector Travel (mm)	Collision Switches	Percept Collision
Right Reach 1	12	52	728.08	0 1 0 0 0 0 0	Lego_toy
Right Reach 2	32	83	1448.84	0 1 1 0 0 0 0	Lego_toy
Left Reach 1	14	51	871.25	1 0 0 0 0 0 0	Barney_toy

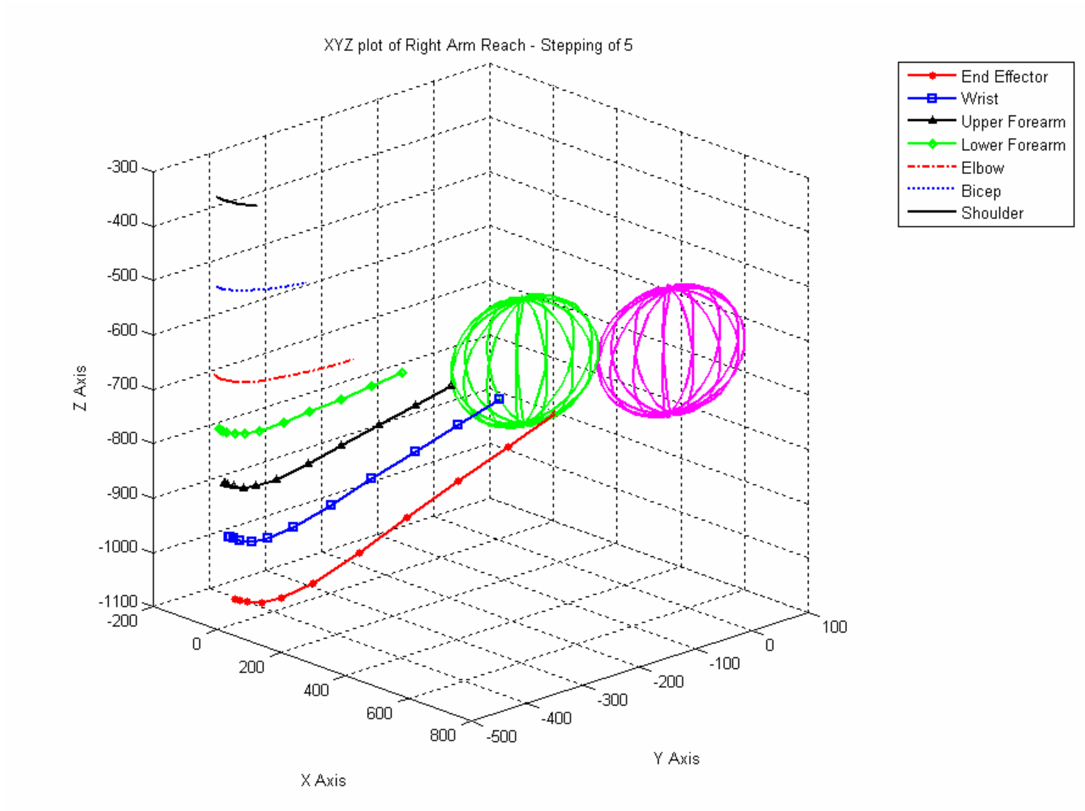


Figure 57: Exp 2 – Right Reach Behavior – Stepping of 5

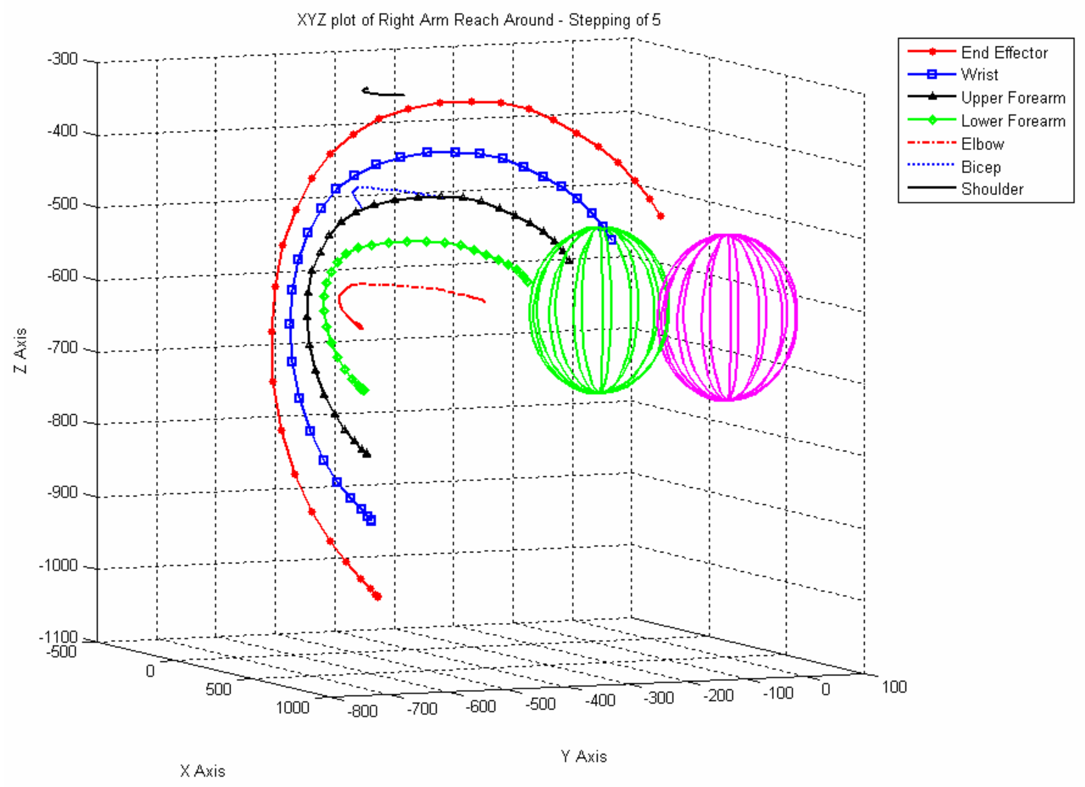


Figure 58: Exp 2 – Right Reach Around Behavior – Stepping of 5

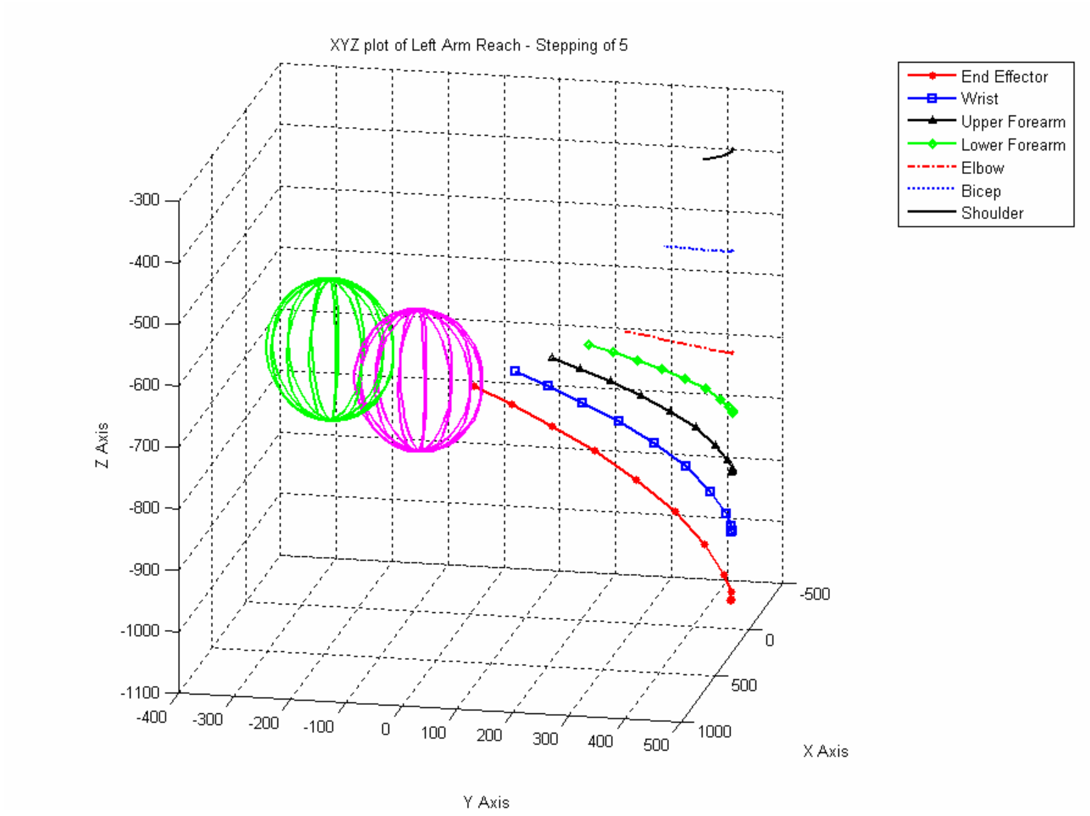


Figure 59: Exp 2 – Left Reach Behavior – Stepping of 5

Discussion of Results

During each of the five trials to test the Internal Rehearsal System, ISAC successfully reached to Barney by performing a regular reach using the left arm. For trials 2-5, IRS behaved in an optimal or best case, manner; trials 2-5 are known as best case scenarios. ISAC was asked to reach to Barney. IRS was then able to complete both of its phases during the arm interpolation phase for all three reach behaviors, and ISAC reached to Barney using its left arm. This entire process for successfully reaching to Barney is shown in figure 60.

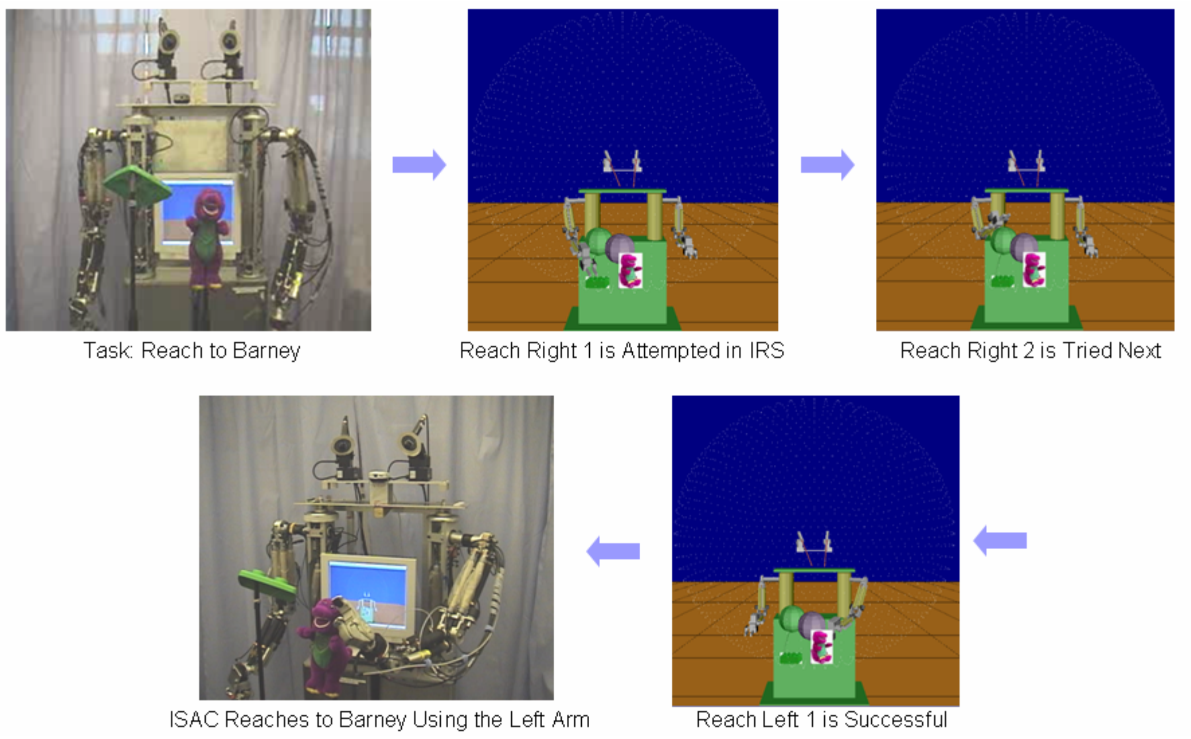

Task: Reach to Barney
Reach Right 1 is Attempted in IRS
Reach Right 2 is Tried Next
Reach Left 1 is Successful
ISAC Reaches to Barney Using the Left Arm

Figure 60: Exp 2 – ISAC Reaches to Barney Using IRS (Best Case Scenario)

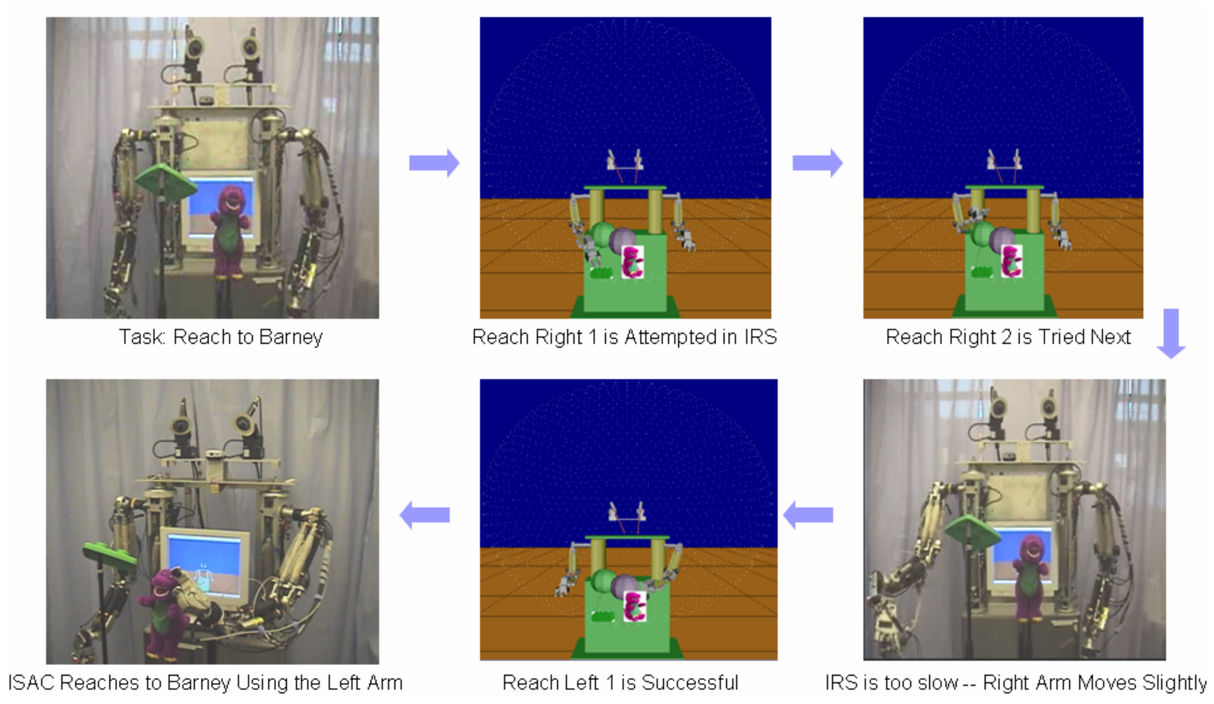

Task: Reach to Barney
Reach Right 1 is Attempted in IRS
Reach Right 2 is Tried Next
IRS is too slow -- Right Arm Moves Slightly
Reach Left 1 is Successful
ISAC Reaches to Barney Using the Left Arm

Figure 61: Exp 2 – ISAC Reaches to Barney Using IRS (Worst Case Scenario)

For trial 1 though, IRS encountered a worst case scenario. ISAC was asked to reach to Barney. IRS was able to complete both its interpolation and motion phases during "ReachRight1" and "ReachLeft1", but it was not able to complete both phases during "ReachRight2". This caused the arm to move somewhat (approximately 3 seconds). Even though the arm moved, no collision occured with the physical Lego toy. ISAC decided to use the left reach behavior because according to IRS, this behavior would successfully accomplish the task. This process can be seen in Figure 61.

However, two main questions came up during this experiment. The first is a question of accuracy, and the second is a question of timing. To answer the accuracy question, figure 62 shows the average end effector travel distance for all five trials. This graph shows that each trial had an average end effector travel distance way below both the radius and diameter of the collision spheres. This means that all of these steppings are acceptable in terms of accuracy, but there is one interesting fact. The reach around behavior distance has a smaller slope ($y = 13.6x - 1.7$) than both of the regular reach behaviors ($y = 9.4x - 0.4$). This is because the reach around behavior is slightly slower than the reach behaviors. Stephen Gordon recorded the regular reach behaviors and Joseph Hall recorded the reach around behavior. This graph shows that even though one behavior is faster than the other, both of them are still far below the diameter line. Therefore, new behaviors recorded for Verbs and Adverbs should be fine with these five steppings.

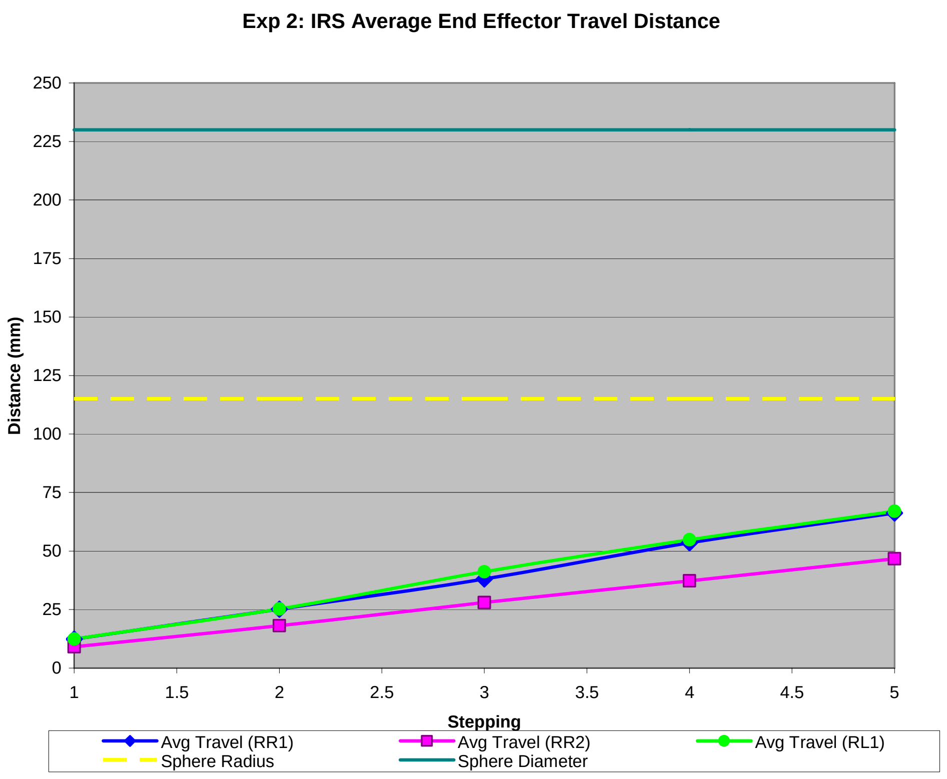


Figure 62: Exp 2 – IRS Average End Effector Travel Distance

All five steppings may be acceptable in terms of accuracy, but the next question is acceptability in terms of time. According to figure 63, the first stepping is unacceptable. This is because IRS timing took longer than than the Arm Agent interpolation time for right reach around creating the worse case scenario. Steppings 2 and 3 are acceptable in terms of timing because IRS is able to compute faster than the Arm Agent; this is seen in that the dashed line is above the solid lines in figure 63 for both cases. Also, steppings 4 and 5 are so fast that the second arm trajectory is not computed (seen as zero timing in the graph). This is because the Arm Agent does not listen to the Activator Agent when it is computing an arm trajectory. This is not necessarily negative; it simply shows that IRS is much much faster than the Arm Agent at this point. So, a stepping of 5 seems to be the best choice from this experiment because it is the fastest and it has acceptable accuracy for all three behaviors.

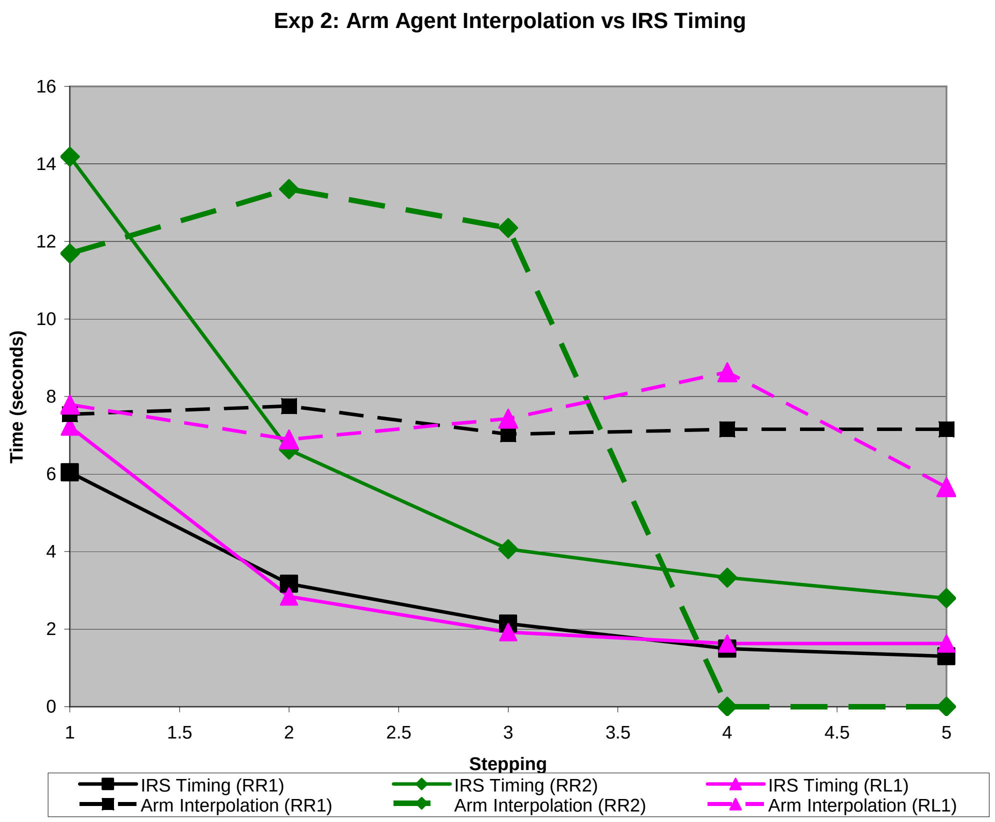


Figure 63: Exp 2 – Arm Agent Interpolation vs IRS Timing

CHAPTER VI

CONCLUSION AND FUTURE WORK

Conclusion

Internal rehearsal is an important feature which allows a cognitive robot to ponder its actions before actually performing them. This thesis showed that the concept of robotic internal simulation, as shown in [Shanahan, 2005], can be expanded from a small mobile robot that has to choose between three cylinders to a more complex, cognitive system like ISAC.

From experiments 1 and 2, a stepping of 5 seems to be the best choice because it is the fastest that IRS can go at this point while having a short average EET distance. However, one interesting question remains. The Arm Agents take about 7 seconds to interpolate an arm trajectory for a regular reach behavior, and it take about 12 seconds to produce a trajectory for a reach around behavior. This is because the Arm Agents use six verb examples to interpolate across. The Arm Agents reside on a computer named Popeye that has a 1.4 Ghz Pentium III processor, Windows 2000 operating system, and 1 gigabyte of RAM. IRS uses the same example files, but it resides on the computer with ISAC's cognitive system. This computer, named Aquateen, has a 3.0 Ghz Pentium D processor, Windows XP operating system, and 3.0Ghz of RAM. This is why IRS is faster than the Arm Agents using a stepping of 1. According to [Spratley, 2006], two factors affect Verb/Adverb interpolation time: number of example files, and the length of the example files. This thesis modified the length of the example files whilst keeping the number constant. This is because if one decides to use fewer example files for the

Arm Agents, IRS can also use fewer example files as well. This should speed up IRS enough to keep up with the faster Arm Agents.

Future Work

Hopefully, IRS system will be improved upon in the future. Much work can be done regarding this research. First of all, IRS currently uses spheres to represent the objects on SES. This is because the objects that Barney interacts with are toys such as Barney, and different colored bean bags. However, in the future, ISAC could use collision shapes other than spheres. Figure 64 shows an example of this concept; in many situations, such as working memory experiments, ISAC manipulates objects on a black table [Gordon and Hall, 2006].

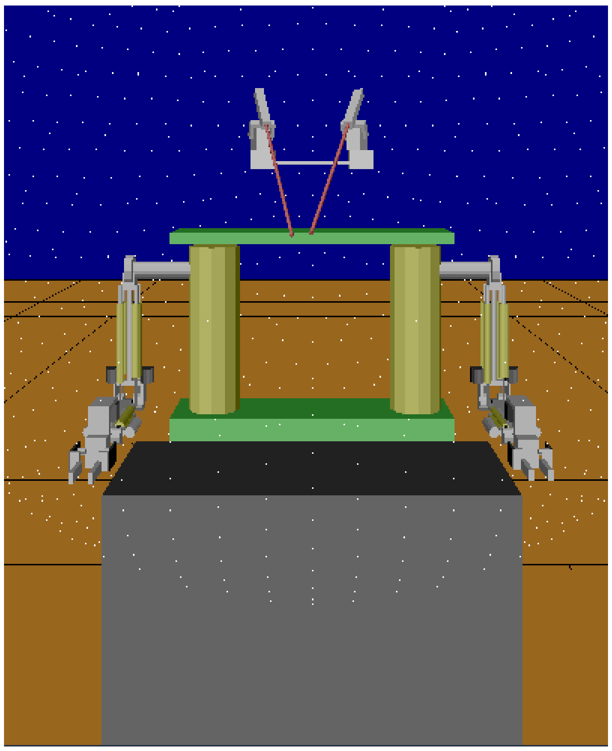


Figure 64: IRS Table Rectangular Collision Box

Currently, ISAC cannot perceive this table with its perceptual agents because these agents are not trained to do so. However, it would be wise to give IRS the ability to represent this table in a different manner than a spherical collision domain. A rectangular box representation would best represent the table. If a collision point on ISAC's arms entered this box, IRS would predict a collision with the table.

Another idea for the Internal Rehearsal System is to allow it to be used as a mechanism to produce new behaviors. For instance, ISAC will be given a task to reach to an object but all the current behaviors produce a collision with an obstacle. ISAC will stop working on the current task and enter a self reflective mode. During this state, ISAC will try to figure out a new behavior to solve the current problem.

Internal rehearsal can also incorporate internal variables such as emotion variables. For instance, will reaching to Barney produce a positive response in ISAC's Affect Agent? IRS may be able to not only predict problems like running into obstacles; it may produce a prediction involving the future of ISAC's internal state.

APPENDIX A

Instruction Manual

Running the Internal Rehearsal System is rather straightforward. This component receives input from the Working Memory System and produces output for the Central Executive Agent. No input is needed from the user, but the Self Agent must be running in order for IRS to function properly. The Self Agent is currently being developed by Palis Ratanaswasd. Below are the instructions for running the Self Agent with internal rehearsal:

1. Log into the computer called “Aquateen” using the Administrator account
2. Go to the folder: “C:\Documents and Settings\Administator\Desktop\Agent Shortcuts
3. Run the ISAC Speech Agent
4. Run the Intention Agent
5. Run the Activator Agent
6. Run the First-Order Response Agent
7. Run the Central Executive Agent
    a. In the Central Executive panel, Click “Populate Search Table”
    b. Next, in the same panel, click “Activate”

8. After these five agents are running, Click the folder “Internal Rehearsal”
    a. This folder has Internal Rehearsal programs at different steppings. IRS with a stepping of 1 is named InternalRehearsalS1; IRS with a stepping of 2 is labeled InternalRehearsalS2 etc...
    b. When IRS is running, a console saying “System Started” will be displayed on the screen. This console will give the user information about how the system is running. (output to CEA, timing etc...)
    c. IRS will save the trajectory created by the Verb/Adverb program in the following folder: “C:\IRSRecord”. A shortcut is located on the desktop.

9. After the programs are running on Aquateen, use the ISAC speech program to give input to the system. This can be accomplished by either speaking into the microphone or by typing words into the field labeled “Whisper a phrase to ISAC”.
    a. ISAC will respond to many commands, but for this thesis, the following commands were the most useful:
        i. Reach to <object> (This will command ISAC to reach to the object. The Self Agent will determine how to reach to the object.)
        ii. Track <object> (This will cause ISAC to saccade to the object and track it using the stereotopic cameras until the stop command is given.)
        iii. Stop all (Stops any command, relaxes the arms, and resets the eyes.)

APPENDIX B

Complete Software File List

Main Files

1. *Collision.cpp and Collision.h* – Used for Collision Detection with virtual spheres
2. *InternalRehearsal.cpp* – Main Executable File
3. *VirtualArm.cpp and VirtualArm.h* – Acts like a virtual arm in the Internal Rehearsal system. This code is originally derived from the Right and Left Arm Agents

Supporting Files

4. *BodyInternal.h and BodyInternal.cpp* – Code helps the virtual arms operate
5. *Location.h and Location.cpp* – Another class used to assist Verb/Adverb code
6. *Matrix.h and Matrix.cpp* – A matrix class used with Verb/Adverb code
7. *Memmap.h and Memmap.cpp* – Memory map related code
8. *Motion*.h and *Motion.cpp* – Used to store example motions for Verb/Adverb code
9. *SAKinematics.h* and *SAKinematics.cpp* – The kinematics class for the virtual arm. (shown in this appendix)
10. *SES.h and SES.cpp* – Used to communicate with the SES
11. *Vbeh.h* and *Vbeh.cpp* – Helper class for the Verb/Adverb class
12. *Verb.h* and *Verb.cpp* – Verb and Adverb code

Main Header Files

This section contains important header files necessary to run the Internal Rehearsal System. Collision.h was written by Joe Hall, and VirtualArm.h was originally written by Stephen Gordon. It was later modified by Joe Hall.

Collision.h

```
#include "SAKinematics.h"
#include <stdio.h>
#include "ses.h"
#include "bodyinternal.h"
#include <vector>
#include <cmath>
#include "Memmap.h"

class Collision
{
public:

		int i,j;
		double desiredangles[6];

		CCreateNewSES ses;
		CSAKinematics Kin;

		double B2x;
		double B2y;
		double B2z;

		double DTR;

		double E2x;
		double E2y;
		double E2z;
		double trans[6],matrix[16];

		CMemMapFile MMF_Collision;

		char *returnedinfo[SES_COL_SIZE];
		char information[SES_COL_SIZE][SES_LIST_SIZE];

		char object[RARM_LIST_SIZE];

		char buffer[1000];
		int colswitch[7];
		int armswitch[7];

		Collision();
		~Collision();
		void InitCollision(int IsLeftArm);
		void CollisionDetection(double arm_angles[6], char perceptgoal[25], int IsLeftArm, int JointStep, int JointSize);
		int CollisionDetectionHelper(double arm_angles[6], char obstaclelist[SES_COL_SIZE][SES_LIST_SIZE], char
*obstacle[SES_COL_SIZE], int IsLeftArm, int jointstep);
		void RecordToFile(double arm_angles[6], FILE* fp);
		void SendToCEA();
		int CheckBuffer();
   int CheckWorkspace(int IsLeftArm, char *goal[RARM_COL_SIZE]);
		void ForwardKinematics(double *j, double *c);
		double dD3;
		double dA2; // These correspond to the lengths for the DH Parameters for the arms
		double dD4;

		double travel;
		int count;

		double xprev;
		double yprev;
		double zprev;

		double xcurr;
		double ycurr;
		double zcurr;

		int offset;

};
```

VirtualArm.h

```
#include "SAKinematics.h"
#include <stdio.h>
#include "ses.h"
#include "bodyinternal.h"
#include <vector>
#include <cmath>
#include "verb.h"
//#include "Memmap.h"
#include "Collision.h"


class VirtualArm
{
public:

		double JointAngles[6];
		double XYZRPY[6];
		CSAKinematics Kin;
		double BaseXForm[3];
		double EndEffXForm[3];

		double dD3;
		double dD4;
		double dA2;

		double DTR;

		int i,j;
		double desiredangles[6];

		CCreateNewSES ses;
		CCreateNewInternalBODY body;

		Collision C;

		char *returnedinfo[SES_COL_SIZE];
		char information[SES_COL_SIZE][SES_LIST_SIZE];

		char object[RARM_LIST_SIZE];

		//char buffer[1000];
		char rightfile[100];
		char leftfile[100];

		int colswitch[7];
```

```
        int armswitch[6];

        int starttime;
        int finishtime;

        int off1;
        int off2;

        VirtualArm();
        ~VirtualArm();
        void FKin(double *joints, double *cart);
        void IKin(double *cart, double *joints);
        void InitVirArm(int IsLeftArm);
        int RunVirRightArm(int rightcall);
        int RunVirLeftArm(int leftcall);
        void AddMotion(char *filename);
        int AddNewVerb(char *name);
        void InitializeVA();
        void GenNewMotion(verb *ActiveVerb, char *motionName, double *advs);
        void create_verb(char *m_szVerbName);
};
```

Main Executable Files

This section contains important executable files necessary to run the Internal Rehearsal System. Collision.cpp was written by Joe Hall, and VirtualArm.cpp was originally written by Stephen Gordon. It was later modified by Joe Hall. InternalRehearsal.cpp is the main file that runs the entire system. It was also written by Joe Hall.

Collision.cpp

```
// Class Collision
// Written by Joe Hall Jan-March 2007

#include <afx.h>
#include <afxwin.h>
#include "stdafx.h"
#include "Collision.h"
#include <winioctl.h>
#include <vector>
#include "verb.h"



Collision::Collision()
{


        strcpy(buffer, "");

        if (!MMF_Collision.MapExistingMemory(_T("MMFCOLLISION"), 500*sizeof(TCHAR)))
        {
                //map the shared memory,
                if (!MMF_Collision.MapMemory(_T("MMFCOLLISION"), 500*sizeof(TCHAR)))
                {
                  printf("Collision Memory Map Failure");
                }

        }


}

Collision::~Collision()
{

}

void Collision::InitCollision(int IsLeftArm)
{

        DTR = 3.14159/180.0;

        dA2=330.0;
        dD3=-200;
        if(IsLeftArm == 1)
                dD3=200;
        dD4=290.0;
```

```
		B2x=-42.0;
		B2y=-265.0;	// Set up Kinematics Class
		if(IsLeftArm == 1)
			B2y=265.0;
		B2z=-330.0;

		E2x=0;
		E2y=0;
		E2z=0;

		Kin.SetParameters(dD3,dD4,dA2,0.0);

		trans[0]=E2x;
		trans[1]=E2y;
		trans[2]=E2z;
		trans[3]=0.0;
		trans[4]=0.0;
		trans[5]=0.0;

		Kin.SetRPYMatrix(trans, matrix);
		Kin.SetW2ETransform(matrix);

		for (i=0; i<7; i++)
			colswitch[i]=0;

		for (i=0; i<7; i++)
			armswitch[i]=0;

		strcpy(buffer, "");

		count = 0;
		travel=0;

		offset=110;

}


void Collision::ForwardKinematics(double *j, double *c)
{
		Kin.SetAngles(j);
		Kin.GetXYZRPY(c);

		c[0]=B2x + c[0];
		c[1]=B2y + c[1];
		c[2]=B2z + c[2];

}

int Collision::CheckWorkspace(int IsLeftArm, char *goal[RARM_COL_SIZE])
{

		char *goalses[SES_COL_SIZE];
		char goalseslist[SES_COL_SIZE][SES_LIST_SIZE];
		double azimuth;
		double elevation;
		double distance;
		double x,y,z;
		double xx,yy,zz;
		double arm_angles[6];
		double coordinates[6];
		double conndist;

		for(int ii=0;ii<SES_COL_SIZE;ii++)
			strcpy(goalseslist[ii], "NULL");

		//strcpy(goalses, goal);

		strcpy(goalseslist[1], goal[RARM_OBJECT]);
```

```
            if( ses.Present(goalseslist) )
   {

                        ses.Retrieve(goalseslist, goalses);

                        azimuth = atof(goalses[SES_AZIMUTH])+90.0;
                        elevation = atof(goalses[SES_ELEVATION]);
                        distance = atof(goalses[SES_DISTANCE]);

                        x= 1000*distance*sin(DTR*elevation)*cos(DTR*azimuth);
                        y= 1000*distance*sin(DTR*elevation)*sin(DTR*azimuth); // Spherical to Polar Coordinates
                        z= 1000*distance*cos(DTR*elevation) ;                  // Actual position

                        xx= y;
                        yy= -x;
                        zz= z+(1655 - 1520);  // 1655 = height of SES origin, 1520 = height of arm origin

                        for(int j=0; j<6; j++)
                        {
                                    arm_angles[j]=0.0;
                        }


   }


            if(IsLeftArm) // Left Arm Origin
            {

                        Kin.SetParameters(0.0,0.0,0.0,0.0);
                        ForwardKinematics(arm_angles, coordinates);

                        conndist= sqrt( (coordinates[0]-xx)*(coordinates[0]-xx) + (coordinates[1]-yy)*(coordinates[1]-yy) + (coordinates[2]-
zz)*(coordinates[2]-zz));

            }
            else    // Right Arm Origin
            {

                        Kin.SetParameters(0.0,0.0,0.0,0.0);
                        ForwardKinematics(arm_angles, coordinates);

                        conndist= sqrt( (coordinates[0]-xx)*(coordinates[0]-xx) + (coordinates[1]-yy)*(coordinates[1]-yy) + (coordinates[2]-
zz)*(coordinates[2]-zz));

            }

            if(conndist < (double)(dD4 + dA2 + 150))
            {
                        printf("Goal percept in Workspace\n");
                        return 0;
            }
            else
            {
                        printf("Goal percept not in Workspace\n");
                        return 1;
            }

            //printf(" %lf %lf %lf Perceptxyz %lf %lf %lf Shoulder\n", xx, yy, zz, coordinates[0], coordinates[1], coordinates[2]);
            //printf(" %lf conndist %lf ArmLength\n", conndist, (double)(dD4 + dA2));

}



// Name: Joe Hall
// Date: March 12, 2007
// Function: RecordToFile
// Purpose: Records the performance of the system, and determines distance traveled by end effector
```

```
void Collision::RecordToFile(double arm_angles[6], FILE* fp)
{

        char *obstacle[SES_COL_SIZE];
        char obstaclelist[SES_COL_SIZE][SES_LIST_SIZE];
        double azimuth;
        double elevation;
        double distance;
        double x,y,z;
        double xx,yy,zz;
        double coordinates[6];


        char string[100];
        int colret=0;

        for(int ii=0;ii<SES_COL_SIZE;ii++)
                strcpy(obstaclelist[ii], "NULL");

        strcpy(obstaclelist[1], "barney_toy"); // barney_toy is Present in the SES, Check for Collision
        if( ses.Present(obstaclelist) )
  {

                ses.Retrieve(obstaclelist, obstacle);

                azimuth = atof(obstacle[SES_AZIMUTH])+90.0;
                elevation = atof(obstacle[SES_ELEVATION]);
                distance = atof(obstacle[SES_DISTANCE]);

                x= 1000*distance*sin(DTR*elevation)*cos(DTR*azimuth);
                y= 1000*distance*sin(DTR*elevation)*sin(DTR*azimuth); // Spherical to Polar Coordinates
                z= 1000*distance*cos(DTR*elevation) ;              // Actual position

                xx= y;
                yy= -x;
                zz= z+(1655 - 1520);  // 1655 = height of SES origin, 1520 = height of arm origin

                fprintf(fp, "%lf %lf %lf ", xx, yy, zz);

  }

        strcpy(obstaclelist[1], "lego_toy"); // barney_toy is Present in the SES, Check for Collision
        if( ses.Present(obstaclelist) )
  {

                ses.Retrieve(obstaclelist, obstacle);

                azimuth = atof(obstacle[SES_AZIMUTH])+90.0;
                elevation = atof(obstacle[SES_ELEVATION]);
                distance = atof(obstacle[SES_DISTANCE]);

                x= 1000*distance*sin(DTR*elevation)*cos(DTR*azimuth);
                y= 1000*distance*sin(DTR*elevation)*sin(DTR*azimuth); // Spherical to Polar Coordinates
                z= 1000*distance*cos(DTR*elevation) ;              // Actual position

                xx= y;
                yy= -x;
                zz= z+(1655 - 1520);  // 1655 = height of SES origin, 1520 = height of arm origin

                fprintf(fp, "%lf %lf %lf ", xx, yy, zz);
  }


        Kin.SetParameters(dD3,dD4+offset,dA2,0.0); // EEffector
        ForwardKinematics(arm_angles, coordinates);
        fprintf(fp, "%lf %lf %lf ", coordinates[0], coordinates[1], coordinates[2]);

  // Also determine distance traveled by Wrist

        if(count == 0)
		{

			xprev=coordinates[0];
			yprev=coordinates[1];
			zprev=coordinates[2];

			xcurr=coordinates[0];
			ycurr=coordinates[1];
			zcurr=coordinates[2];

		}
		else
		{

			xprev=xcurr;
			yprev=ycurr;
			zprev=zcurr;

			xcurr=coordinates[0];
			ycurr=coordinates[1];
			zcurr=coordinates[2];

			travel+=sqrt((xcurr-xprev)*(xcurr-xprev) + (ycurr-yprev)*(ycurr-yprev) + (zcurr-zprev)*(zcurr-zprev));

			//printf("travel %lf\n", travel);

			//printf("xyzprev %lf %lf %lf xyzcurr %lf %lf %lf\n", xprev, yprev, zprev, xcurr, ycurr, zcurr);

		}

		count++;


		Kin.SetParameters(dD3,dD4,dA2,0.0); // wrist
		ForwardKinematics(arm_angles, coordinates);
		fprintf(fp, "%lf %lf %lf ", coordinates[0], coordinates[1], coordinates[2]);

		Kin.SetParameters(dD3,(2*dD4)/3,dA2,0.0); // upper forearm
		ForwardKinematics(arm_angles, coordinates);
		fprintf(fp, "%lf %lf %lf ", coordinates[0], coordinates[1], coordinates[2]);

		Kin.SetParameters(dD3,(1*dD4)/3,dA2,0.0); // lower forearm
		ForwardKinematics(arm_angles, coordinates);
		fprintf(fp, "%lf %lf %lf ", coordinates[0], coordinates[1], coordinates[2]);

		Kin.SetParameters(dD3,0.0,dA2,0.0); // elbow
		ForwardKinematics(arm_angles, coordinates);
		fprintf(fp, "%lf %lf %lf ", coordinates[0], coordinates[1], coordinates[2]);

		Kin.SetParameters(dD3,0.0,dA2/2,0.0); // bicep
		ForwardKinematics(arm_angles, coordinates);
		fprintf(fp, "%lf %lf %lf ", coordinates[0], coordinates[1], coordinates[2]);

		Kin.SetParameters(dD3,0.0,0.0,0.0); // shoulder
		ForwardKinematics(arm_angles, coordinates);
		fprintf(fp, "%lf %lf %lf\n", coordinates[0], coordinates[1], coordinates[2]);


		Kin.SetParameters(dD3,dD4,dA2,0.0); // Reset Values

}



// Name: Joe Hall
// Date: Feb 20, 2007
// Function: Collision Detection
// Purpose: This function reads which percepts are located on the SES and determines
// their coordinates. If a percept exists, the collisiondetectionhelper function is called
// to determine if a collision has occured with the operating arm.

void Collision::CollisionDetection(double arm_angles[6], char perceptgoal[25], int IsLeftArm, int JointStep, int JointSize) //Just compare the
distances between objects
{
                                        //Bring the SES list in

            char *obstacle[SES_COL_SIZE];
            char obstaclelist[SES_COL_SIZE][SES_LIST_SIZE];
            char string[100];
            int colret=0;


            //printf("Percept goal %s \n", perceptgoal);
            //printf("Getting into Collision Detection\n");
           //printf("%lf %lf %lf %lf %lf %lf\n", arm_angles[0]*DTR, arm_angles[1]*DTR, arm_angles[2]*DTR, arm_angles[3]*DTR,
arm_angles[4]*DTR, arm_angles[5]*DTR);

               for(int ii=0;ii<SES_COL_SIZE;ii++)
                                      strcpy(obstaclelist[ii], "NULL");

      strcpy(obstaclelist[1], "red_bag"); // Red Bag is Present in the SES, Check for Collision
      if( ses.Present(obstaclelist)  ) //&& !(strcmp("red_bag", goal))
      {
                                       //printf("Red Bag is Present \n");
                                      colret=0;
                                      ses.Retrieve(obstaclelist, obstacle);
                                      colret= CollisionDetectionHelper(arm_angles, obstaclelist, obstacle, IsLeftArm, JointStep);


                                      if ((colswitch[0] == 0) && (colret != 0))
                                      {

                                              sprintf(string, "%d %d %lf %s [%d %d %d %d %d %d %d]\n", JointStep, JointSize, travel,
obstacle[SES_NAME], armswitch[0], armswitch[1], armswitch[2], armswitch[3], armswitch[4], armswitch[5], armswitch[6]);
                                              strcat(buffer, string);
                                              colswitch[0]=1;

                                      }

      }

                              strcpy(obstaclelist[1], "blue_bag"); // Blue Bag is Present in the SES, Check for Collision
      if( ses.Present(obstaclelist))
      {
                                      colret=0;
                                      ses.Retrieve(obstaclelist, obstacle);
                                      colret=CollisionDetectionHelper(arm_angles, obstaclelist, obstacle, IsLeftArm, JointStep);


                                      if ((colswitch[1] == 0) && (colret != 0))
                                      {

                                              sprintf(string, "%d %d %lf %s [%d %d %d %d %d %d %d]\n", JointStep, JointSize, travel,
obstacle[SES_NAME], armswitch[0], armswitch[1], armswitch[2], armswitch[3], armswitch[4], armswitch[5], armswitch[6]);
                                              strcat(buffer, string);
                                              colswitch[1]=1;

                                      }

      }

                              strcpy(obstaclelist[1], "barney_toy"); // Barney is Present in the SES, Check for Collision
      if( ses.Present(obstaclelist) )
      {
                                      colret=0;
                                      ses.Retrieve(obstaclelist, obstacle);
                                      colret=CollisionDetectionHelper(arm_angles, obstaclelist, obstacle, IsLeftArm, JointStep);

                                      if ((colswitch[2] == 0) && (colret != 0))
```

```
                                        {

                                                sprintf(string, "%d %d %lf %s [%d %d %d %d %d %d %d]\n", JointStep, JointSize, travel,
obstacle[SES_NAME], armswitch[0], armswitch[1], armswitch[2], armswitch[3], armswitch[4], armswitch[5], armswitch[6]);
                                                strcat(buffer, string);
                                                colswitch[2]=1;

                                        }

        }

                        strcpy(obstaclelist[1], "pear_toy"); // Pear is Present in the SES, Check for Collision
        if( ses.Present(obstaclelist) )
        {
                                        colret=0;
                                        ses.Retrieve(obstaclelist, obstacle);
                                        colret=CollisionDetectionHelper(arm_angles, obstaclelist, obstacle, IsLeftArm, JointStep);

                                        if ((colswitch[3] == 0) && (colret != 0))
                                        {

                                                sprintf(string, "%d %d %lf %s [%d %d %d %d %d %d %d]\n", JointStep, JointSize, travel,
obstacle[SES_NAME], armswitch[0], armswitch[1], armswitch[2], armswitch[3], armswitch[4], armswitch[5], armswitch[6]);
                                                strcat(buffer, string);
                                                colswitch[3]=1;

                                        }

        }

                        strcpy(obstaclelist[1], "lego_toy"); // Lego is Present in the SES, Check for Collision
        if( ses.Present(obstaclelist) )
        {
                                        colret=0;
                                        ses.Retrieve(obstaclelist, obstacle);
                                        colret=CollisionDetectionHelper(arm_angles, obstaclelist, obstacle, IsLeftArm, JointStep);

                                        if ((colswitch[4] == 0) && (colret != 0))
                                        {

                                                sprintf(string, "%d %d %lf %s [%d %d %d %d %d %d %d]\n", JointStep, JointSize, travel,
obstacle[SES_NAME], armswitch[0], armswitch[1], armswitch[2], armswitch[3], armswitch[4], armswitch[5], armswitch[6]);
                                                strcat(buffer, string);
                                                colswitch[4]=1;

                                        }


        }

                        strcpy(obstaclelist[1], "car_toy"); // Car Toy is Present in the SES, Check for Collision
        if( ses.Present(obstaclelist) )
        {
                                        colret=0;
                                        ses.Retrieve(obstaclelist, obstacle);
                                        colret=CollisionDetectionHelper(arm_angles, obstaclelist, obstacle, IsLeftArm, JointStep);

                                        if ((colswitch[5] == 0) && (colret != 0))
                                        {

                                                sprintf(string, "%d %d %lf %s [%d %d %d %d %d %d %d]\n", JointStep, JointSize, travel,
obstacle[SES_NAME], armswitch[0], armswitch[1], armswitch[2], armswitch[3], armswitch[4], armswitch[5], armswitch[6]);
                                                strcat(buffer, string);
                                                colswitch[5]=1;

                                        }

        }
```

```
                    strcpy(obstaclelist[1], "flamingo_toy"); // Flamingo is Present in the SES, Check for Collision
        if( ses.Present(obstaclelist) )
        {
                            colret=0;
                            ses.Retrieve(obstaclelist, obstacle);
                            colret=CollisionDetectionHelper(arm_angles, obstaclelist, obstacle, IsLeftArm, JointStep);

                            if ((colswitch[6] == 0) && (colret != 0))
                            {

                                    sprintf(string, "%d %d %lf %s [%d %d %d %d %d %d %d]\n", JointStep, JointSize, travel,
obstacle[SES_NAME], armswitch[0], armswitch[1], armswitch[2], armswitch[3], armswitch[4], armswitch[5], armswitch[6]);
                                    strcat(buffer, string);
                                    colswitch[6]=1;

                            }

        }


}


// Name: Joe Hall
// Date: Feb 20, 2007
// Function: CollisionDetectionHelper
// Purpose: This function determines if a percept is within a certain threshold
// value of the wrist, forearm, elbow, bicep, shoulder and end effector. This function
// Does the actual hard work.
int Collision::CollisionDetectionHelper(double arm_angles[6], char obstaclelist[SES_COL_SIZE][SES_LIST_SIZE], char
*obstacle[SES_COL_SIZE], int IsLeftArm, int jointstep )
{

        double conndist=0;
        double xx, yy, zz;
        double x, y, z;
        double threshold = 115;
        double coordinates[6];

        double azimuth;
        double elevation;
        double distance;
        int returnval = 0;


  azimuth = atof(obstacle[SES_AZIMUTH])+90.0;
  elevation = atof(obstacle[SES_ELEVATION]);
        distance = atof(obstacle[SES_DISTANCE]);

  x= 1000*distance*sin(DTR*elevation)*cos(DTR*azimuth);
  y= 1000*distance*sin(DTR*elevation)*sin(DTR*azimuth); // Spherical to Polar Coordinates
  z= 1000*distance*cos(DTR*elevation) ;                 // Actual position

        xx= y;
        yy= -x;
        zz= z+(1655 - 1520);  // 1655 = height of SES origin, 1520 = height of arm origin

        Kin.SetParameters(dD3,dD4+offset,dA2,0.0); // EEffector
        ForwardKinematics(arm_angles, coordinates);
        conndist= sqrt( (coordinates[0]-xx)*(coordinates[0]-xx) + (coordinates[1]-yy)*(coordinates[1]-yy) + (coordinates[2]-
zz)*(coordinates[2]-zz));
        if((conndist < threshold) && (armswitch[0] == 0))
        {

                if (IsLeftArm == 1)
                        printf("Collision with left end effector with %s\n", obstacle[SES_NAME]);
                else
                        printf("Collision with right end effector with %s\n", obstacle[SES_NAME]);

                armswitch[0]=1;
```

```
			returnval = 1;

		}


		Kin.SetParameters(dD3,dD4,dA2,0.0); // wrist
		ForwardKinematics(arm_angles, coordinates);

		conndist= sqrt( (coordinates[0]-xx)*(coordinates[0]-xx) + (coordinates[1]-yy)*(coordinates[1]-yy) + (coordinates[2]-
zz)*(coordinates[2]-zz));
		//printf("wrist %lf\n", conndist);

		if( (conndist < threshold) && (armswitch[1] == 0) )
		{

			if (IsLeftArm == 1)
				printf("Collision with left wrist with %s\n", obstacle[SES_NAME]);
			else
				printf("Collision with right wrist with %s\n", obstacle[SES_NAME]);


			armswitch[1]=1;
			returnval = 2;

		}


		Kin.SetParameters(dD3,(2*dD4)/3,dA2,0.0); // upper forearm
		ForwardKinematics(arm_angles, coordinates);
		conndist= sqrt( (coordinates[0]-xx)*(coordinates[0]-xx) + (coordinates[1]-yy)*(coordinates[1]-yy) + (coordinates[2]-
zz)*(coordinates[2]-zz));
		//printf("forearm %lf\n", conndist);

		if( (conndist < threshold) && (armswitch[2] == 0) )
		{

			if (IsLeftArm == 1)
				printf("Collision with left upper forearm with %s\n", obstacle[SES_NAME]);
			else
				printf("Collision with right upper forearm with %s\n", obstacle[SES_NAME]);


			armswitch[2]=1;
			returnval = 3;

		}

		Kin.SetParameters(dD3,dD4/3,dA2,0.0); // lower forearm
		ForwardKinematics(arm_angles, coordinates);
		conndist= sqrt( (coordinates[0]-xx)*(coordinates[0]-xx) + (coordinates[1]-yy)*(coordinates[1]-yy) + (coordinates[2]-
zz)*(coordinates[2]-zz));
		//printf("forearm %lf\n", conndist);

		if( (conndist < threshold) && (armswitch[2] == 0) )
		{

			if (IsLeftArm == 1)
				printf("Collision with left lower forearm with %s\n", obstacle[SES_NAME]);
			else
				printf("Collision with right lower forearm with %s\n", obstacle[SES_NAME]);


			armswitch[3]=1;
			returnval = 4;

		}

		Kin.SetParameters(dD3,0.0,dA2,0.0); // elbow
		ForwardKinematics(arm_angles, coordinates);
		conndist= sqrt( (coordinates[0]-xx)*(coordinates[0]-xx) + (coordinates[1]-yy)*(coordinates[1]-yy) + (coordinates[2]-
zz)*(coordinates[2]-zz));
		if( (conndist < threshold) && (armswitch[3] == 0) )
		{

			if (IsLeftArm == 1)
				printf("Collision with left elbow with %s\n", obstacle[SES_NAME]);
			else
				printf("Collision with right elbow with %s\n", obstacle[SES_NAME]);

			armswitch[4]=1;
			returnval = 5;

		}

		Kin.SetParameters(dD3,0.0,dA2/2,0.0); // bicep
		ForwardKinematics(arm_angles, coordinates);
		conndist= sqrt( (coordinates[0]-xx)*(coordinates[0]-xx) + (coordinates[1]-yy)*(coordinates[1]-yy) + (coordinates[2]-
zz)*(coordinates[2]-zz));
		if( (conndist < threshold) && (armswitch[4] == 0) )
		{

			if (IsLeftArm == 1)
				printf("Collision with left bicep with %s\n", obstacle[SES_NAME]);
			else
				printf("Collision with right bicep with %s\n", obstacle[SES_NAME]);


			armswitch[5]=1;
			returnval = 6;
		}

		Kin.SetParameters(dD3,0.0,0.0,0.0); // shoulder
		ForwardKinematics(arm_angles, coordinates);
		conndist= sqrt( (coordinates[0]-xx)*(coordinates[0]-xx) + (coordinates[1]-yy)*(coordinates[1]-yy) + (coordinates[2]-
zz)*(coordinates[2]-zz));
		if( (conndist < threshold) && (armswitch[5] == 0) )
		{

			if (IsLeftArm == 1)
				printf("Collision with left shoulder with %s\n", obstacle[SES_NAME]);
			else
				printf("Collision with right shoulder with %s\n", obstacle[SES_NAME]);


			armswitch[6]=1;
			returnval = 7;
		}


		Kin.SetParameters(dD3,dD4,dA2,0.0); // wrist reset

		return(returnval);

}




// Name: Joe Hall
// Date: March 18, 2007
// Function: Send to CEA
// Purpose: Sends collision information to the CEA.
void Collision::SendToCEA()
{

		printf("%s", buffer);

		if( strcmp(buffer, "") == 0)
		{
			printf("NO collision occured during behavior\n");
			sprintf(buffer, "%d %d %d [0 0 0 0 0 0]\n", -1, -1, -1);
		}


		LPVOID lpBuffer;
		lpBuffer = MMF_Collision.Open(); // post this information to shared memory

		//printf("lpBuffer %d\n", lpBuffer);

		if (lpBuffer)
		{
			_tcscpy((TCHAR*)lpBuffer, buffer);
		}

		MMF_Collision.Close();

}

// Name: Joe Hall
// Date: March 18, 2007
// Function: CheckBuffer
// Purpose: Determines if a collision has occured.

int Collision::CheckBuffer()
{
		int finished = 0;

			if( strcmp(buffer, "") != 0)
			{
				printf("Collision Occured, Finished\n");
				finished = 1;
			}

		return finished;
}
```

InternalRehearsal.cpp

```
#include "stdafx.h"
#include <stdio.h>
#include <vector>
#include "bodyinternal.h"
#include "verb.h"
#include "VirtualArm.h"
#include <time.h>


// Name: Joe Hall
// Date: Feb 20, 2007
// Function: Main Internal Rehearsal Function
// Purpose: This function checks the rarminternal and larminternal mysql database tables
// for commands. If a command is given to either the right or left virtual arm, the arm
// executes the command. Only one arm can operate at a time.

int main(int argc, char* argv[])
{


		VirtualArm V;

		int rightcall=1;
		int leftcall=1;
```

```
	char rightarm[RARM_COL_SIZE][RARM_LIST_SIZE];
	char rightarmupdate[RARM_COL_SIZE][RARM_LIST_SIZE];
	char leftarm[LARM_COL_SIZE][LARM_LIST_SIZE];
	char leftarmupdate[LARM_COL_SIZE][LARM_LIST_SIZE];
	char sesinfo[SES_COL_SIZE][SES_LIST_SIZE];
	int i;

	clock_t starttime, finishtime;

	char buffer[1000];
	char behavior[RARM_LIST_SIZE];
	char percept[RARM_LIST_SIZE];

	CCreateNewSES sesmain;
	CCreateNewInternalBODY bodyint;

	printf("System Started\n");


	for(;;)
	{

		for(i=0;i<RARM_COL_SIZE;i++)
		{
			strcpy(rightarm[i],"NULL");
			strcpy(rightarmupdate[i],"NULL");
		}


		strcpy(rightarm[RARM_ACKNOWLEDGE1], "1");

		if(bodyint.PresentRa(rightarm))
		{
			starttime=clock();
			V.RunVirRightArm(rightcall);
			finishtime=clock();
			printf("Time %d\n\n", (finishtime-starttime));
			Sleep(1000);
			rightcall++;

		}

		for(i=0;i<LARM_COL_SIZE;i++)
		{
			strcpy(leftarm[i],"NULL");
			strcpy(leftarmupdate[i],"NULL");
		}

		strcpy(leftarm[LARM_ACKNOWLEDGE1], "1");

		if(bodyint.PresentLa(leftarm))
		{

			starttime=clock();
			V.RunVirLeftArm(leftcall);
			finishtime=clock();
			printf("Time %d\n\n", (finishtime-starttime));
			Sleep(1000);
			leftcall++;
		}
		Sleep(100);

	}

	return 0;

}
```

VirtualArm.cpp

```
// Class Virtual Arm
// Code Originally Written by Stephen Gordan and others for Right and Left Arm Agents
// Modified by Joe Hall Jan-March 2007

#include <afx.h>
#include <afxwin.h>
#include "stdafx.h"
#include "VirtualArm.h"
#include <winioctl.h>
#include <vector>
#include "Vbeh.h"
//#include "Collision.h"

#define DEFAULT_SAMPLING_PERIOD 0.050
#define NUM_BEHS 2

double DTR = 3.14159/180.0;

//#define SAMPLING_PERIOD 40 // 40 milliseconds
//#define CHANGE_TIME 7000 //500 milliseconds

//using namespace std;

bool GO = true;//false;
bool K = false;

vector<double>joint1;
vector<double>joint2;
vector<double>joint3;
vector<double>joint4;
vector<double>joint5;
vector<double>joint6;

char behavior_file[NUM_BEHS][100];
int AdvCount[NUM_BEHS];

vector<CVbeh> Behs;


/*

        #define DEFAULT_DOF 6
        #define TOTAL_VERBS 5

        #define SAMPLING_PERIOD 40 // 40 milliseconds
        #define CHANGE_TIME 7000 //500 milliseconds

        bool GO = true;//false;
        bool K = false;

        verb *m_pActiveVerb;
        int m_nNumVerbs = 0; // Constants associated with Verbs/Adverbs
        verb **m_VerbList;
        int m_nNumObjects;
        InterpMotion **m_IMotionList;
        int m_nNumIMotion;
        int m_nNumAdverbs = 3;
        int m_nNumKeytimes = 2;

        vector<double>joint1;
        vector<double>joint2;
        vector<double>joint3;
        vector<double>joint4;
        vector<double>joint5;
        vector<double>joint6;
*/
```

```
VirtualArm::VirtualArm()
{

}

VirtualArm::~VirtualArm()
{

}

// Init Vir Arm is used to initialize either the right or left virtual arm
// in the state predictor class. It has some code from the ISACArm.cpp constructor.

void VirtualArm::InitVirArm(int IsLeftArm)
{

		printf("Starting\n");

		DTR = 3.14159/180.0;

		BaseXForm[0]=-42.0;
		BaseXForm[1]=-265.0;
		if(IsLeftArm == 1)
			BaseXForm[1]=265.0;
		BaseXForm[2]=-330.0;

		EndEffXForm[0]=0; // -180.0;
		EndEffXForm[1]=0; //-50;
		EndEffXForm[2]=0.0;

		dA2=330.0;
		dD3=-200;
		if(IsLeftArm == 1)
			dD3=200;
		dD4=290.0;

		Kin.SetParameters(dD3,dD4,dA2,0.0);

		double t[6],m[16];
		for(i=0;i<3;i++)
			t[i]=EndEffXForm[i];
		for(i=3;i<6;i++)
			t[i]=0.0;

		Kin.SetRPYMatrix(t, m);
		Kin.SetW2ETransform(m);


		// Add behavior files below
		strcpy(behavior_file[0], "behavior1.txt");
		AdvCount[0] = 3;

		strcpy(behavior_file[1], "behavior2.txt");
		AdvCount[1] = 3;

}


// Modified ISACarm function
void VirtualArm::IKin(double *cart, double *joints)
{
		cart[0]-=BaseXForm[0];
		cart[1]-=BaseXForm[1];
		cart[2]-=BaseXForm[2];
		Kin.SetXYZRPY(cart);
		Kin.GetAngles(joints);
}
```

```
// Modified ISACarm function
void VirtualArm::FKin(double *joints, double *cart)
{
        Kin.SetAngles(joints);
        Kin.GetXYZRPY(cart);
        cart[0]+=BaseXForm[0];
        cart[1]+=BaseXForm[1];
        cart[2]+=BaseXForm[2];

}


// Verbs/Adverbs Related code "InitializeVA"
// This code assists the right and left arm agent code in
// building verb/adverb interpolation trajectories

void VirtualArm::InitializeVA() // Updated
{
        printf("Initializing Verbs\n");

        for(int ii=0;ii<NUM_BEHS;ii++)
        {

                FILE *fp = fopen(behavior_file[ii],"r");
                int n = AdvCount[ii];

                if(fp)
                {
                        CVbeh vb;
                        vb.m_pActiveVerb = NULL;
                        vb.m_VerbList=NULL;
                        vb.m_nNumVerbs=0;
                        vb.m_nNumObjects=0;
                        vb.m_IMotionList=NULL;
                        vb.m_nNumIMotion=0;


                        if(vb.m_VerbList==NULL)
                                vb.m_VerbList=new verb* [TOTAL_VERBS];

                        char name[100];
                        char buf[100];
                        double* raw_data;
                        double key;
                        double limits[2];

                        raw_data = (double*)malloc(n*sizeof(double));

                        fscanf(fp, "%s", name);
                        fscanf(fp, "%lf", &key);

                        vb.SetAdverbNum(n);
                        vb.SetKeyTimeNum(key);

                        vb.create_verb(name);
//                      vb.AddSyn("reach");

                //      printf("VERB name %s\n", name);
                        for(int jj=0;jj<n;jj++)
                        {
                                fscanf(fp, "%lf", &limits[0]);
                                fscanf(fp, "%lf", &limits[1]);
                                vb.SetAdverbLimits(jj, limits);
                        //      printf("Limits: %d : %lf %lf\n", jj, limits[0], limits[1]);
                        }


                        fscanf(fp, "%s", buf);
                //      printf("Should be adding syns: %s\n", buf);
                        //printf("%s\n", name);
```

```
			while(1)
			{
				if(!strcmp(name, buf))
					break;
				//printf("Adding Synonym: %s to verb %s\n", buf, name);
				vb.AddSyn(buf);
				if(fscanf(fp, "%s", buf) == EOF)
					printf("Error in loading behaviors\n");
			}

			while(fscanf(fp, "%s", buf) != EOF)
			{
				//buf is motion filename
				//printf("Motion name %s ", buf);
				for(jj=0;jj<n;jj++)
				{
					double temp;
					fscanf(fp, "%lf", &temp);
					raw_data[jj] = temp;
					//printf(" with adv %f", raw_data[jj]);
				}
			//	printf("\n");
				vb.AddMotion(buf, raw_data);
			}
			free(raw_data);
			Behs.push_back(vb);
		}
		else
			printf("Can't open file\n");
	}


	printf("Done adding Verbs\n");



}

// Verbs/Adverbs Related code "GenNewMotion"
// This code assists the right and left arm agent code in
// building verb/adverb interpolation trajectories

void VirtualArm::GenNewMotion(verb *ActiveVerb, char *motionName, double *advs) // updated
{
	printf("Generating MOTION!\n");
	double *dpAdvPos;
	int NumExamples;
	int NumDOF;
	int NumAdverbs;
	int NumKeytimes;
	int i, cnt;
	int count = 0;
	int stepping = 1;


	if(ActiveVerb != NULL)
	{
		NumExamples = ActiveVerb->get_NumExamples();
		NumDOF		= ActiveVerb->get_NumDOF();
		NumAdverbs = ActiveVerb->get_NumAdverbs();
		NumKeytimes = ActiveVerb->get_NumKeytimes();

		//printf("%d %d %d\n", NumExamples, NumAdverbs, NumKeytimes);

		cnt=0;
		for(i=0;i<NumExamples;i++)
			if(ActiveVerb->get_ExemplarDetails(i)->Status=='A')
				cnt++;

		dpAdvPos=new double [NumAdverbs];
```

```
				for(i=0;i<NumAdverbs;i++)
				{
					dpAdvPos[i] = advs[i];
					//printf(" %lf ", dpAdvPos[i]);
				}
				//printf("\n");

				//not going to use this... set it myself.
				//CInterpolationDlg dlg(m_pActiveVerb,dpAdvPos); // get adverb space position from user's dialog box
				if(1)
				{
					InterpMotion *im=new InterpMotion(dpAdvPos, NumAdverbs, ActiveVerb->get_NumKeytimes());
					printf("Just interpolated new motion\n");
					//m_pDoc->AddNewInterpMotion(im);
					//Do something with interpolated motion

					ActiveVerb->GenNewMotion(im,DEFAULT_SAMPLING_PERIOD,dpAdvPos);

					//printf("got here 1\n");
					int len, col, j;
					double *data[7];

					len=im->NumDataPoints();

					col=ActiveVerb->get_NumDOF()+1;

					//printf("%d %d\n", len, col);
					for(i=0;i<col;i++)  //  allocate memory for joint angle data
						data[i]=new double [len];

					for(j=0; j<len; j++)  // read joint angle data from file
						for(i=0; i<col; i++)
							data[i][j]=im->get_data(j,i);

					//printf("got here\n");
					FILE *fp;
					fp = fopen(motionName,"w");
					fprintf(fp, "%d %d\n", col, len);
					for(j=0;j<len;j++)
					{


						if (count == stepping)
						{
							joint1.push_back(data[1][j]);
							joint2.push_back(data[2][j]);
							joint3.push_back(data[3][j]);
							joint4.push_back(data[4][j]);
							joint5.push_back(data[5][j]);
							joint6.push_back(data[6][j]);

							for(i=0;i<col;i++)
							{
								fprintf(fp, "%lf ", data[i][j]);
							}
							fprintf(fp, "\n");

							count = 0;
						}
						count++;
					}
				}
				delete []dpAdvPos;
			}
}

int VirtualArm::RunVirRightArm(int rightcall)
{

			InitVirArm(0); // Initialize Right Arm
```

```
		FILE* fp;

		C.InitCollision(0); // Right Arm

		strcpy(rightfile, "");
		sprintf(rightfile, "C:/IRSRecord/RightArm%d.txt", rightcall);

// Start of Right Arm Agent Code (uses verbs/adverbs interpolation)

		InitializeVA();

		//fp = fopen("C:/IRSRecord/RightArm.txt", "w");  // Open file for writing
		fp = fopen(rightfile, "w");  // Open file for writing

		int ii;
		char info[SES_COL_SIZE][SES_LIST_SIZE];
		char info_update[SES_COL_SIZE][SES_LIST_SIZE];

		char rarm_info[RARM_COL_SIZE][RARM_LIST_SIZE];
		char rarm_update[RARM_COL_SIZE][RARM_LIST_SIZE];

		for(ii=0;ii<SES_COL_SIZE;ii++)
		{
			strcpy(info[ii], "NULL");
			strcpy(info_update[ii], "NULL");
		}

		for(ii=0;ii<RARM_COL_SIZE;ii++)
		{
			strcpy(rarm_info[ii], "NULL");
			strcpy(rarm_update[ii], "NULL");
		}

		int i;
		int index = 0;
		int finished = 0;
		int started = 0;

		double jts[6];

		jts[0] = 0.0;
		jts[1] = 1.57;
		jts[2] = -3.14;
		jts[3] = 0.0;
		jts[4] = 0.0;
		jts[5] = 0.0;

		body.PutArmJtsRa(jts);

		//printf("GOING TO STOP\n");
		//getchar();
		//return 1;

		DWORD timeelapsed;
		DWORD motiontime;
		bool NODE_REACH = false;
		bool OBJ_REACH = false;
		bool FOUND_BEH = false;
		bool MOVE_HOME = false;
		bool HANDSHAKE = false;
		bool STOP = false;
		bool INTERPOLATED = false;

		int node = -1;
		int count = 0;
		while(finished == 0)
		{

			for(ii=0;ii<RARM_COL_SIZE;ii++)
				strcpy(rarm_info[ii],"NULL");
```

```
                strcpy(rarm_info[RARM_ID], "1");
                strcpy(rarm_update[RARM_ID], "1");

                if(body.PresentRa(rarm_info))
                {
                        CVbeh current_beh;
                        char *ret_info[RARM_COL_SIZE];
                        char beh[RARM_LIST_SIZE];
                        char obj[RARM_LIST_SIZE];

                        if(body.RetrieveRa(rarm_info, ret_info))
                        {

                                int ack1 = (int)(atof(ret_info[RARM_ACKNOWLEDGE1]));
                                if(ack1 == 1 && started == 0)
                                {
                                        started = 1;
                                        joint1.clear();
                                        joint2.clear();
                                        joint3.clear();
                                        joint4.clear();
                                        joint5.clear();
                                        joint6.clear();
                                        count = 0;
                                        INTERPOLATED = false;
                                        strcpy(beh, ret_info[RARM_BEHAVIOR]);
                                        strcpy(obj, ret_info[RARM_OBJECT]);

                                        printf("got some command %s\n", beh);
                                        DWORD sometime1 = GetTickCount();
                                                //printf("Received Command at Time %d\n", sometime1);
                                        starttime=clock();

                                        if(!strcmp("reach1", beh))
                                        {
                                                off1=10;
                                                off2=45;
                                        }
                                        else
                                        {
                                                off1=10;
                                                off2=37;

                                        }


                                        int size = Behs.size();
                                        for(int b=0;b<size;b++)
                                        {
                                                CVbeh tBeh = Behs.at(b);
                                                for(int s=0;s<tBeh.m_nSynNum;s++)
                                                {
                                                //              printf("%s %s\n", tBeh.syn[s], beh);
                                                        if(!strcmp(tBeh.syn[s], beh))
                                                        {
                                                        //              printf("We have a winner! %s && %s\n", tBeh.syn[s],
        beh);

                                                                current_beh = tBeh;
                                                                b = size;
                                                                s = 25;
                                                                FOUND_BEH = true;
                                                        }
                                                }
                                        }
                                        if(!FOUND_BEH)
                                        {
                                                if(!strcmp(beh, "Stop"))
                                                {
                                                        printf("Got command to STOP\n");
                                                        FOUND_BEH = false;
                                                        STOP = true;
                                                }
                                        }

                                        strcpy(rarm_update[RARM_ACKNOWLEDGE1], "0");
                                        body.UpdateRa(rarm_info, rarm_update);

                                }
                        }

                        if(FOUND_BEH)
                        {
                                if(!INTERPOLATED)
                                {
                                        joint1.clear();
                                        joint2.clear();
                                        joint3.clear();
                                        joint4.clear();
                                        joint5.clear();
                                        joint6.clear();
                                        //Sleep(1500);
                                        char *ret_infoses[SES_COL_SIZE];
                                        for(ii=0;ii<SES_COL_SIZE;ii++)
                                                strcpy(info[ii], "NULL");
                                        strcpy(info[SES_NAME], obj);

                                        ses.Retrieve(info, ret_infoses);
                                        double range = atof(ret_infoses[SES_DISTANCE]);
                                        double azimuth = atof(ret_infoses[SES_AZIMUTH]);
                                        double elevation = atof(ret_infoses[SES_ELEVATION]);
                                        double vals[3] = {azimuth, elevation, range};
                                        int flag = 3;
                                        current_beh.CalcAdverbs(vals, flag);
                                        //printf("Trying to interpolate\n");

                                        int num_advs = current_beh.current_advs.size();
                                        //printf("ADVERB size %d\n", current_beh.current_advs.size());
                                        double advs[20];
                                        for(i=0;i<current_beh.m_nNumAdverbs;i++)
                                                advs[i] = current_beh.current_advs.at(i);

                                        GenNewMotion(current_beh.m_VerbList[0], "motion", advs);
                                        //GenNewMotion(m_VerbList[index], "RMotion", advs);
                                        INTERPOLATED = true;
                                        int ss=joint1.size();
                                        //printf("Joint Size %d\n", ss);
                                        DWORD sometime = GetTickCount();

                                        finishtime=clock();
                                        printf("Interpolation Time %d\n", (finishtime-starttime));

                                        starttime=clock();


                                }
                                bool WRITEARM = true;
                                int size = joint1.size();
                                if(count >= size-1)
                                {
                                        printf("Finished object reach: MOTION\n");

                                        WRITEARM = false;
                                        FOUND_BEH = false;
                                        finished = 1;
                                }
                                double jt1 = joint1.at(count);
                                double jt2 = joint2.at(count);
                                double jt3 = joint3.at(count);
                                double jt4 = joint4.at(count);
                                double jt5 = joint5.at(count);
```

```
                        double jt6 = joint6.at(count);
                        count++;

                        double des_ang[6] = {jt1, jt2, jt3, jt4, jt5, jt6};

                        timeelapsed = GetTickCount();

                        if(WRITEARM)
                        {
                                des_ang[0]=des_ang[0] + off1*DTR;
                                des_ang[2]=des_ang[2] - off2*DTR;

                                body.PutArmJtsRa(des_ang); // Run Right Virtual Arm
                                for(int k=0; k<6; k++)
                                        desiredangles[k]=des_ang[k];

                        }

                }


        }
        else
                body.InsertRa(rarm_info);

        // End of Right Arm Agent Code

        // My code starts here

        C.CollisionDetection(desiredangles, object, 0, count, joint1.size());

        C.RecordToFile(desiredangles, fp); // Record Arm and Percept data to file

        if(C.CheckBuffer())// Break from the loop if a collision occurs
        {
                finished = 1;
                //printf("Finished = %d\n", finished);

                finishtime=clock();
                printf("Collision Time %d\n", (finishtime-starttime));

        }

    } // end of while loop

    C.SendToCEA();

    jts[0] = 0.0;
    jts[1] = 1.57;
    jts[2] = -3.14;
    jts[3] = 0.0;
    jts[4] = 0.0;
    jts[5] = 0.0;

    body.PutArmJtsRa(jts);
    fclose(fp);

// My code ends here
    return 1;
}


int VirtualArm::RunVirLeftArm(int leftcall)
{

    InitVirArm(1); // Initialize Left Arm
    FILE* fp;

    C.InitCollision(1); // Left Arm
```

```
                strcpy(leftfile, "");
                sprintf(leftfile, "C:/IRSRecord/LeftArm%d.txt", leftcall);

// Start of Left Arm Agent Code (uses verbs/adverbs interpolation)

                InitializeVA();

                //fp = fopen("C:/IRSRecord/RightArm.txt", "w");  // Open file for writing
                fp = fopen(leftfile, "w");  // Open file for writing

                int ii;
                char info[SES_COL_SIZE][SES_LIST_SIZE];
                char info_update[SES_COL_SIZE][SES_LIST_SIZE];

                char larm_info[LARM_COL_SIZE][LARM_LIST_SIZE];
                char larm_update[LARM_COL_SIZE][LARM_LIST_SIZE];

                for(ii=0;ii<SES_COL_SIZE;ii++)
                {
                                strcpy(info[ii], "NULL");
                                strcpy(info_update[ii], "NULL");
                }

                for(ii=0;ii<LARM_COL_SIZE;ii++)
                {
                                strcpy(larm_info[ii], "NULL");
                                strcpy(larm_update[ii], "NULL");
                }

                int i;
                int index = 0;
                int finished = 0;
                int started = 0;

                double jts[6];

                jts[0] = 0.0;
                jts[1] = 1.57;
                jts[2] = -3.14;
                jts[3] = 0.0;
                jts[4] = 0.0;
                jts[5] = 0.0;

                body.PutArmJtsLa(jts);

                //printf("GOING TO STOP\n");
                //getchar();
                //return 1;

                DWORD timeelapsed;
                //DWORD motiontime;
                bool NODE_REACH = false;
                bool OBJ_REACH = false;
                bool FOUND_BEH = false;
                bool MOVE_HOME = false;
                bool HANDSHAKE = false;
                bool STOP = false;
                bool INTERPOLATED = false;

                int node = -1;
                int count = 0;
                while(finished == 0)
                {

                                for(ii=0;ii<LARM_COL_SIZE;ii++)
                                                strcpy(larm_info[ii],"NULL");
                                strcpy(larm_info[LARM_ID], "1");
                                strcpy(larm_update[LARM_ID], "1");

                                if(body.PresentRa(larm_info))
```

```
{
        CVbeh current_beh;
        char *ret_info[LARM_COL_SIZE];
        char beh[LARM_LIST_SIZE];
        char obj[LARM_LIST_SIZE];

        if(body.RetrieveLa(larm_info, ret_info))
        {

                int ack1 = (int)(atof(ret_info[LARM_ACKNOWLEDGE1]));
                if(ack1 == 1 && started == 0)
                {

                        started = 1;
                        joint1.clear();
                        joint2.clear();
                        joint3.clear();
                        joint4.clear();
                        joint5.clear();
                        joint6.clear();
                        count = 0;
                        INTERPOLATED = false;
                        strcpy(beh, ret_info[LARM_BEHAVIOR]);
                        strcpy(obj, ret_info[LARM_OBJECT]);

                        printf("got some command %s\n", beh);
                        DWORD sometime1 = GetTickCount();
                        //      printf("Received Command at Time %d\n", sometime1);

                        starttime=clock();

                        if(!strcmp("reach1", beh))
                        {
                                off1=10;
                                off2=45;
                        }
                        else
                        {
                                off1=10;
                                off2=37;

                        }


                        int size = Behs.size();
                        for(int b=0;b<size;b++)
                        {
                                CVbeh tBeh = Behs.at(b);
                                for(int s=0;s<tBeh.m_nSynNum;s++)
                                {
                                        //printf("%s %s\n", tBeh.syn[s], beh);
                                        if(!strcmp(tBeh.syn[s], beh))
                                        {
                                        //printf("We have a winner! %s && %s\n", tBeh.syn[s], beh);
                                                current_beh = tBeh;
                                                b = size;
                                                s = 25;
                                                FOUND_BEH = true;
                                        }
                                }
                        }
                        if(!FOUND_BEH)
                        {
                                if(!strcmp(beh, "Stop"))
                                {
                                        printf("Got command to STOP\n");
                                        FOUND_BEH = false;
                                        STOP = true;
                                }
                        }
```

```
				strcpy(larm_update[LARM_ACKNOWLEDGE1], "0");
				body.UpdateLa(larm_info, larm_update);

			}
		}

		if(FOUND_BEH)
		{
			if(!INTERPOLATED)
			{
				joint1.clear();
				joint2.clear();
				joint3.clear();
				joint4.clear();
				joint5.clear();
				joint6.clear();
				//Sleep(1500);
				char *ret_infoses[SES_COL_SIZE];
				for(ii=0;ii<SES_COL_SIZE;ii++)
					strcpy(info[ii], "NULL");
				strcpy(info[SES_NAME], obj);

				ses.Retrieve(info, ret_infoses);
				double range = atof(ret_infoses[SES_DISTANCE]);
				double azimuth = -atof(ret_infoses[SES_AZIMUTH]);
				double elevation = atof(ret_infoses[SES_ELEVATION]);
				double vals[3] = {azimuth, elevation, range};
				int flag = 3;
				current_beh.CalcAdverbs(vals, flag);
				//printf("Trying to interpolate\n");

				int num_advs = current_beh.current_advs.size();
			//	printf("ADVERB size %d\n", current_beh.current_advs.size());
				double advs[20];
				for(i=0;i<current_beh.m_nNumAdverbs;i++)
					advs[i] = current_beh.current_advs.at(i);

				GenNewMotion(current_beh.m_VerbList[0], "motion", advs);
				//GenNewMotion(m_VerbList[index], "RMotion", advs);
				INTERPOLATED = true;
				int ss=joint1.size();
				//printf("Joint Size %d\n", ss);
				DWORD sometime = GetTickCount();
				//printf("Finished interpolation at time = %d\n", sometime);

				finishtime=clock();
				printf("Interpolation Time %d\n", (finishtime-starttime));

				starttime=clock();

			}
			bool WRITEARM = true;
			int size = joint1.size();
			if(count >= size-1)
			{
				printf("Finished object reach: MOTION\n");

				WRITEARM = false;
				FOUND_BEH = false;
			}
			double jt1 = joint1.at(count);
			double jt2 = joint2.at(count);
			double jt3 = joint3.at(count);
			double jt4 = joint4.at(count);
			double jt5 = joint5.at(count);
			double jt6 = joint6.at(count);
			count++;

			double des_ang[6] = {jt1, jt2, jt3, jt4, jt5, jt6};
```

```
                        timeelapsed = GetTickCount();

                        if(WRITEARM)
                        {
                                des_ang[0]=-des_ang[0] - off1*DTR;
                                des_ang[2]=des_ang[2] - off2*DTR;

                                body.PutArmJtsLa(des_ang); // Run Right Virtual Arm
                                for(int k=0; k<6; k++)
                                        desiredangles[k]=des_ang[k];

                        }

                }


        }
        else
                body.InsertLa(larm_info);

        // End of Right Arm Agent Code

        // My code starts here

        C.CollisionDetection(desiredangles, object, 1, count, joint1.size());

        C.RecordToFile(desiredangles, fp); // Record Arm and Percept data to file

        if(C.CheckBuffer())// Break from the loop if a collision occurs
        {
                finished = 1;
                //printf("Finished = %d\n", finished);
                finishtime=clock();
                printf("Collision Time %d\n", (finishtime-starttime));

        }

    } // end of while loop

    C.SendToCEA();

    jts[0] = 0.0;
    jts[1] = 1.57;
    jts[2] = -3.14;
    jts[3] = 0.0;
    jts[4] = 0.0;
    jts[5] = 0.0;

    body.PutArmJtsLa(jts);
    fclose(fp);


    return 1;
}
```

BIBLIOGRAPHY

Achim K., *Image Mapping and Visual Attention on a Sensory Ego-Sphere*, Master's Thesis, Vanderbilt University, August 2005.

Albus, J. S. “Outline for a Theory of Intelligence.” *IEEE Trans. Syst., Man, Cybern.,* vol 21, no. 3, pp. 473-509, 1991.

Alford, W.A., *Design and Implementation of the Self Agent in a Multiagent-Based Robot Control Architecture*, Ph.D. Dissertation, Nashville, TN: Vanderbilt University, 2000.

Anderson, J. R. “ACT: A Simple Theory of Complex Cognition”. *American Psychologist*, 51, 355-365. 1996.

Anderson, J. R., et al., “An Integrated Theory of the Mind,” *Psychological Review 111,* (4). pp.1036-1060, 2004.

Baars, BJ., “In the Theatre of Consciousness: Global Workspace Theory, A Rigorous Scientific Theory of Consciousness”, *Journal of Consciousness Studies*, 4, No. 4, 1997, pp. 292-309.

Baddeley, A.D. *Working Memory*, Clarendon Press, Oxford, UK. 1986.

Baddeley, A.D. & Hitch, G.J., *Working Memory*, In G.A. Bower (Ed.), *Recent Advances in Learning and Motivation*, Vol. 8, pp. 47-90, Academic Press, New York, NY. 1974.

Bridgestone Corporation, “Rubbertuators and Applications for Robotics, Technical Guide No. 1”, 1986.

Brooks, R. A., Breazeal (Ferrell), C., Irie, R., Kemp, C. C., Marjanović, M., Scassellati, B., and Williamson, M. M. “Alternative Essences of Intelligence”. In *Proceedings of the Fifteenth National/Tenth Conference on Artificial intelligence/innovative Applications of Artificial intelligence* (Madison, Wisconsin, United States). American Association for Artificial Intelligence, Menlo Park, CA, 961-968. 1998.

Brooks, R.A., *COG*, http://www.ai.mit.edu/projects/humanoid-robotics-group/cog/ (accessed June 17, 2007)

Brooks, R.A. “Intelligence without Representation”, *Artificial Intelligence,* vol.47, no.1-3, pp.139-160. 1991.

Budiu, R., *About ACT-R*, http://act-r.psy.cmu.edu/about/ (accessed Feb 1, 2007).

Camperi, M. and Wang, X. “A model of visuospatial working memory in prefrontal cortex: Recurrent network and cellular bistability”. *Journal of Computational Neuroscience*, 5:383–405, 1998.

Charoenseang, S. “A PC-Based Virtual Reality System For Dual-Arm Humanoid Robot Control”, Ph.D. Dissertation, Vanderbilt University, 1999.

Christopher, J.L., *A PneuHand for Human-like Grasping on a Humanoid Robot*, Master’s Thesis, Vanderbilt University, Nashville, TN, 1998.

DARPA, *Cognitive Information Processing Technology*, DARPA-BAA 02-21, June 2002.

Dearden A. and Demiris Y., “Learning Forward Models for Robots”, in: Proceedings of International Joint Conference on Artificial Intelligence (IJCAI), 2005, pp. 1440-1445.

Dodd, W., *The Design of Procedural, Semantic, and Episodic Memory Systems for a Cognitive Robot*, Master’s Thesis, Vanderbilt University, August 2005.

Dodd, W., and Gutierrez, R.., “Episodic Memory and Emotion for Cognitive Robot**,”** *14th IEEE Int’l Workshop on Robot and Human Interactive Communication (RO-MAN)*, Nashville, TN, Aug 13-15, 2005.

Erol, D., et al., “Motion generation for humanoid robots with automatically derived behaviors,” *Proc. of IEEE Int’l. Conf. on Systems, Man, and Cybernetics,* Washington, DC, Oct. 6-8, 2003, pp. 1816-1821, 2003.

Franklin, S., “Autonomous Agents as Embodied AI,” *Cybernetics and Systems Special Issue on Epistemological Aspects of Embodied AI*, 28:6 499-520, 1997.

Franklin, S. and A. Graesser, “Is It an Agent, or Just a Program? A Taxonomy for Autonomous Agents,” *Intelligent Agents III: Agent Theories Architectures, and Languages*, J. Mueller, Ed., Berlin: Springer, 1997.

Franklin, S., A. Kelemen, and L. McCauley, “IDA: A Cognitive Agent Architecture,” *IEEE Conf on Systems, Man and Cybernetics*, 1998.

Gordon, S. & Hall, J. “System Integration with Working Memory Management for Robotic Behavior Learning”, *ICDL*, Bloomington, IN. 2006.

Haikonen, P., *The Cognitive Approach to Conscious Machines*. Exeter, UK: Imprint Academic, 2003.

Hesslow, G. “Conscious thought as simulation of behavior and perception”. *Trends in Cognitive Science*, 6(6). 242-247. 2002.

Hopfstadter, D.R. and Mitchell, M., “The Copycat Project: A model of mental fluidity and analogy-making”, *Advances in Connectionist and Neural Computation Theory, Vol. 2: Logical Connections*, eds. K.J. Holyoak and J.A. Barnden, Norwood, NJ: Ablex, 1994.

Kawamura, K., Dodd, D., Ratanaswasd, P., and Gutierrez, R. A., “Development of a robot with a sense of self,” *The 6th IEEE Int’l Symposium on Computational Intelligence in Robotics and Automation (CIRA)*, Espoo, Finland, June 21-30, 2005.

Kawamura, K. and Gordon, S., “From Intelligent Control to Cognitive Control”. In the 11th International Symposium on Robotics and Applications (ISORA) 2006.

Kawamura, K., Gordon, S., Ratanaswasd P., (2007a) “Robotic Body-Mind Integration: Next Grand Challenge in Robotics”, *Industrial Robotics Theory, Modelling & Control*, pp. 1-42, January 2007.

Kawamura, K., Gordon, S., Ratanaswasd, P., Erdemir, E., Hall, J., (2007b) “Implementation of Cognitive Control for a Humanoid Robot ”, submitted to *International Journal of Humanoid Robotics*, June 2007, Under Review.

Kawamura, K., Gordon, S., Ratanaswasd, P., Garber, C., Erdemir, E., (2007c) “Implementation of Cognitive Control for Robots”, *Proceedings of the 4th COE Workshop on Human Adaptive Mechatronics (HAM)*, March 2007.

Kawamura, K., Noelle, D.C., Hambuchen, K.A. and Rogers, T.E. “A Multi-agent Approach to Self-reflection for Cognitive Robots", *Proc. of 11th Int’l Conf. on Advanced Robotics*, pp. 568-575, Coimbra, Portugal, 2003.

Kawamura, K., Peters, R. A., Bagchi, S., Iskarous, M., and Bishay, M. "Intelligent robotic systems in service of the disabled", *IEEE Transactions on Rehabilitation Engineering*, vol. 3, no. 1, pp. 14-21, March, 1995.

Laird, J. E., Newell, A., and Rosenbloom, P. S., "SOAR: An Architecture for General Intelligence", *Artificial Intelligence 33,* pp. 1-64, 1987.

Maes, P., “How to do the Right Thing”, *Connection Science*, 1, pp. 291-323.

Miller, E. K., “The Prefrontal Cortex and Cognitive Control”, *Nature Reviews: Neuroscience*, 2000.

Miller, E. K.; Erickson, C. A. and Desimone, R., “Neural Mechanisms of Visual Working Memory in Prefrontal Cortex of the Macaque”, *Jo. of Neuroscience,* vol.16, pp.5154-6. 1996.

Miller, G. A. “The Magical Number Seven, Plus or Minus two: Some Limits on Our Capacity for Processing Information”. *Psychological Review*, 63:81–97, 1956.

Minsky, M., *The Society of Mind*, New York: Simon and Schuster, 1985.

Olivares, R.E. *Intelligent Machine Architecture 1.0: Introduction and system overview,* CIS /Cognitive Robotics Laboratory Technical Report, Vanderbilt University, Nashville, TN, May 2003.

Olivares, R.E. *The Intelligent Machine Architecture 2.5: A revised development environment and software architecture,* Master Thesis, Vanderbilt University, Nashville, TN, May 2004.

Pack, R.T. *IMA: The Intelligent Machine Architecture*, PhD Dissertation, Vanderbilt University, Nashville, TN, May 1998.

Peters, R.A. II, Hambuchen, K.A., Kawamura, K., Wilkes, D.M. “The Sensory Ego- Sphere as a Short-Term memory for Humanoids”. *Proceedings of the IEEE-RAS Conference on Humanoid Robots*, 2001, pp 451-60.

Phillips, J. and Noelle, D. “A Biologically Inspired Working Memory Framework for Robots”, *14th IEEE Int’l Workshop on Robot and Human Interactive Communication*, Nashville, TN, Aug 13-15, 2005, pp 599-601.

Phillips, J. and Noelle, D. “Working Memory for Robots: Inspirations from Computational Neuroscience”, *Proc. from 5th Int’l Conf on Development and Learning,* Bloomington, IN, June 2006.

Ratanaswasd, P., Dodd, W., Kawamura, K., and Noelle, D., “Modular Behavior Control for a Cognitive Robot”, *12th International Conference on Advanced Robotics (ICAR*), Seattle, Washington, 2005.

Rojas, J., *Sensory Integration with Articulated Motion on a Humanoid Robot,* Master's Thesis, Nashville, TN: Vanderbilt University, May 2004.

Rose, C., Cohen, M.F., and Bodenheimer, R., "Verbs and Adverbs: Multidimensional motion interpolation," *IEEE Computer Graphics and Applications,* vol. 18, no. 5, Sep.-Oct. 1998, pp.32-40.

Schroder, J. *Humanoid Robot Arm actuated by Artificial Muscles,* Master's Thesis, Nashville, TN: Vanderbilt University, May 2003.

Shanahan, M. "A Cognitive Architecture that Combines Internal Simulation with a Global Workspace", Available ScienceDirect.com, In Press, Received April 2005.

Shrobe, H.E et al., "CHIP: A Cognitive Architecture for Comprehensive Human Intelligence and Performance," September 27, 2006.

Skubic, M., et al. "A Biologically Inspired Adaptive Working Memory for Robots,"2004 *AAAI Symposium Series*, Washington, D.C., October 21-24, 2004.

Spratley II, A.W., *Verbs and Adverbs as the Basis for Motion Generation in Humanoid Robots,* Master's Thesis, Vanderbilt University, Nashville, TN, August 2006.

Sun, G. and Scassellati, B., "Reaching through learned forward model", in *IEEE-RAS/RSJ International Conference on Humanoid Robots*. Santa Monica, CA: IEEE, 2004.

Sutton, R. S., "Learning to predict by the methods of temporal differences". *Machine Learning*, 3:9–44, 1988.

Vikenmark, D. and Minguez, J., "Reactive Obstacle Avoidance for Mobile Robots that Operate in Confined 3D Workspaces," IEEE MELCON 2006, May 16-19, Benalmadena, pp. 1246-1251.

Wolpert, D. M., and Kawato, M., "Multiple Paired Forward and Inverse Models for Motor Control", *Neural Networks,* 11, 1317–1329, 1998.

Wray, R., Chong, R., Phillips, J., Rogers, S., and Walsh, B., *A Survey of Cognitive and Agent Architectures*, http://ai.eecs.umich.edu/cogarch0/ (accessed Feb 1, 2007).