Cognitive Rovio: Using RovioWrap and .NET to Control a Rovio

Scott Settembre
Department of Computer Science and Engineering, University at Buffalo
CSE 736 : Seminar on Cognitive Robotics
ss424@cse.buffalo.edu

Abstract

I address the issue of cognitive control of a robot by implementing the GLAIR architecture in a .NET environment. Firstly, I control the basic functions of a robot and give it primitive behaviors. Secondly, I approach the issue of cognitive control by examining what it means to be cognitive for a robot and what would be necessary to implement such a system. Finally, I develop an application implemented as a finite state machine to realize some of these requirements and execute the equivalent of a simple cognitive algorithm.

Contents

1. Introduction

The controlling of a robot is not a simple task and requiring an additional restriction that the control must be ‘cognitive’ makes the task even harder. The AAAI 1998 Fall Symposium on Cognitive Robotics places a clear definition on Cognitive Robotics:

> “Research in Cognitive Robotics is concerned with the theory and the implementation of robots that reason, act and perceive in changing, incompletely known, unpredictable environments. Such robots must have higher level cognitive functions that involve reasoning, for example, about goals, actions, when to perceive and what to look for, the cognitive states of other agents, time, collaborative task execution, etc. In short, Cognitive Robotics is concerned with integrating reasoning, perception and action within a uniform theoretical and implementation framework.”

In particular, it is not enough to have perception and action, but an integrated reasoning strategy must be employed. So from this definition, we can come up with some guidelines for a cognitive robot. Thus, a cognitive robot must satisfy the following principles:

I. Developed with a uniform theoretical and implementation framework
II. Have higher level cognitive functions that involve reasoning
III. Reasoning must be about or include:
    a. Goals
    b. Actions

c. When to perceive and what to look for

d. Cognitive states of itself and other agents

e. Time

Over the course of the semester, I have come up with several questions regarding what makes something cognitive. What makes a normal, sequential program, not a cognitive program? Why is it that a **while** loop in C is not considered cognitive even if the code within has a behavior is identical to that of a series of instructions that is considered cognitive? And since any programmed system can be reduced to C code, isn't then that a C program can be cognitive? This and other questions will be explored and hopefully answered throughout this paper.

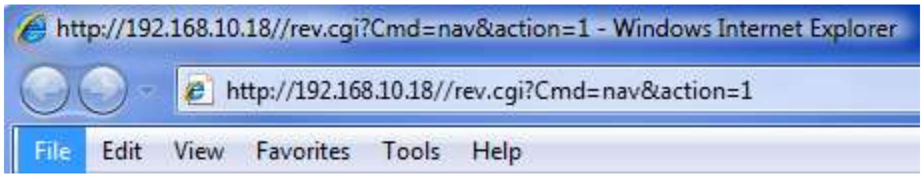
2. Rovio Platform

Dr. Shapiro suggested several robotic platforms that we could use to explore cognitive robotics. I had become familiar with many different types of robotic kits over the years and suggested that one of the latest WowWee products could be adapted to serve the purposes of this course. I quickly set out to prove that it would satisfy the course requirements and spent a lot of time before the course started to develop a wrapper object to communicate to the Rovio.

2.1. RovioWrap

The RovioWrap object, developed in .NET, encapsulates the WowWee Rovio ver 1.2 API. [WowWee 2008] Rovio acts as a mobile web server and can be connected to through any web browser or http web object to call specific cgi scripts (present already on the Rovio) to return various pieces of information from the Rovio sensors or as a request for Rovio to manipulate the wheels or other actuators accessible to the API.

For example, to access the state of all the sensors and settings of the Rovio at IP address 192.168.10.18 , you need only open a web browser and type in:



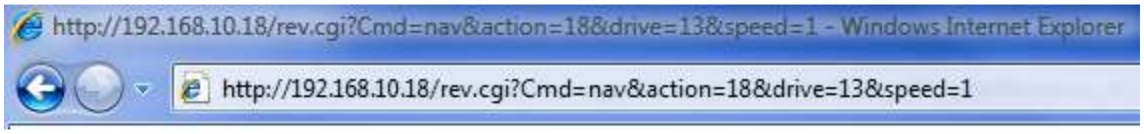
```
http://192.168.10.18//rev.cgi?Cmd=nav&action=1 - Windows Internet Explorer
http://192.168.10.18//rev.cgi?Cmd=nav&action=1
```

which would return the following information:

```
Cmd = nav
responses = 0|x=7923|y=2555|theta=-1.437|room=0|ss=26794
|beacon=0|beacon_x=0|next_room=1|next_room_ss=289
|state=0|resistance=0|sm=15|pp=0|flags=0004
|brightness=3|resolution=1|video_compression=1|frame_rate=30
|privilege=0|user_check=0|speaker_volume=3|mic_volume=5
|wifi_ss=221|show_time=0|ddns_state=0|email_state=0
|battery=114|charging=80|head_position=203|ac_freq=2
```

Out of this stream of data, one can simply parse out the exact bits of information that are needed. For example, if you wish to determine whether an obstacle has been detected, merely parse the above feed to see if the sensor data for field “obstacle=” is either “004” (no obstacle detected) or “006” (obstacle detected).

Should one wish to tell the robot to perform a specific action, for example, to raise its camera to mid level height, one needs only open a web browser and type in:



```
http://192.168.10.18/rev.cgi?Cmd=nav&action=18&drive=13&speed=1 - Windows Internet Explorer
http://192.168.10.18/rev.cgi?Cmd=nav&action=18&drive=13&speed=1
```

At this point the Rovio would immediately raise (or lower) the camera head to middle height.

I personally was quite impressed that the entire API was callable through HTTP commands, which made interfacing with the Rovio nearly completely platform independent. From the experiences of the other students in our class, we were able to use this API regardless of operating system (Mac, Windows, Linux) and also varied programming languages (VB.NET, C#, Java, Python, LISP). This is an excellent model for robotic platforms to use and I highly encourage robotic kit manufacturers to follow suit.

2.2. Open Source Rovio

In the spirit of open source, WowWee exposed their entire API for Rovio to the robotic community. I first became aware of all the interesting things that one could do with the Rovio through the WowWee robotic community forums called RoboCommunity. By scanning the message boards, I was able to determine that I could implement a lot of what was being discussed and prototyped into one nice .NET object.

The result of my efforts has been encapsulated into yet another open source project which I released in early January 2009, called RovioWrap. It is hosted by CodePlex and can be accessed by going to http://www.codeplex.com/roviowrap/ . In the source archives I placed a highly commented code base from which to learn from or start from, including comments outlining where various techniques came from and credits given to forum members of RoboCommunity to thank them for their selfless efforts.

In addition, I have released several videos onto YouTube.com to show the RovioWrap (and the application RovioController) in action. These videos can be accessed by either searching for RovioWrap in google.com or YouTube.com, or going to the open source web page at codeplex.com. There is a video outlining what this project is about, a second video showing some simple obstacle avoidance in the robotic "hello world" equivalent called the "wander" behavior, and a third video that demonstrates the ability to have Rovio understand spoken commands and act on them.

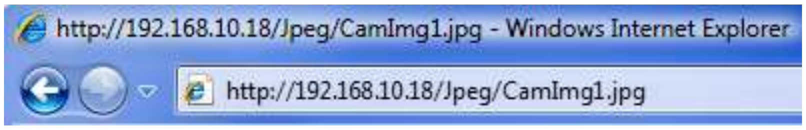
3. Rovio Advanced Behaviors

I set out to implement a series of behaviors and abilities into RovioWrap and RovioController to not only explore the capabilities and limitations of the Rovio Platform, but to pursue the goal of making Rovio a cognitive robot. I used the full capabilities of robot as well as the widely used, rapid-prototyping, .NET programming language to my advantage to implement several very difficult behaviors in a small amount of time. I interfaced and used several additional software packages and leveraged .NET as the coding glue to bring each packages strengths and specialized capabilities together, including Microsoft Speech Recognition and MATLAB.

3.1. Vision

One of my first tasks was to utilize the vision system of the Rovio. There are two ways to access the camera data. The first way is to open up a RSTP [MSDN 2009] connection to Rovio and then have Rovio send a stream of .mpg data using the RTP protocol. This would also require the ability to display and or manipulate the .mpg in a real time fashion, and it was not completely clear if this was necessary to get the data that was needed. The second way was to just request a "snapshot" of the scene every twentieth of a second, or when required, and process each frame individually. Since much of vision processing is done frame by frame, and the techniques I knew would benefit from the already segmented video stream, I decided to use the second technique.

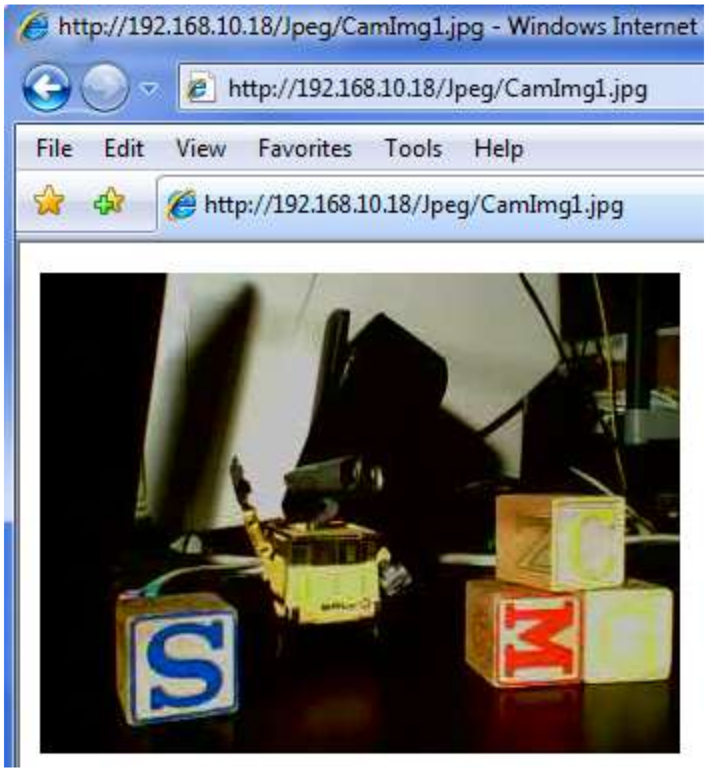
A camera image from Rovio can be requested by calling the cgi script like so:



http://192.168.10.18/Jpeg/CamImg1.jpg - Windows Internet Explorer

http://192.168.10.18/Jpeg/CamImg1.jpg

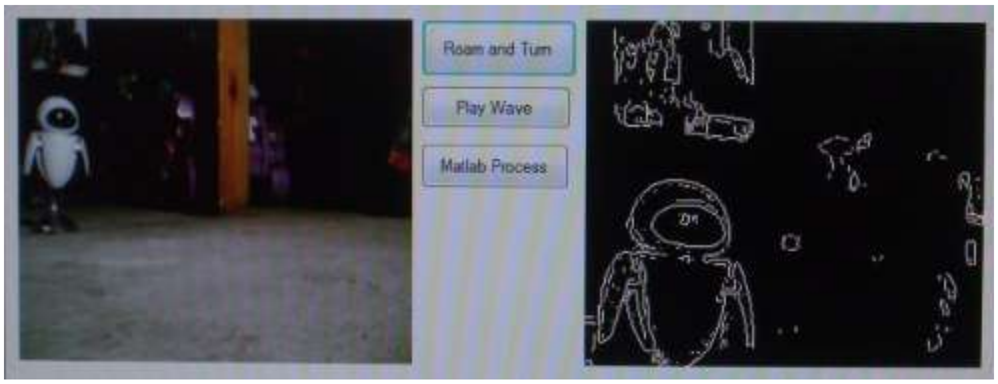
And this will return a .jpg in the return http data. If you happen to be in a web browser, then an image as shown in the following image will be displayed. Otherwise, your web object would need to capture the incoming mime body and place it directly in a file (or a stream) to be used as a .jpg image.


http://192.168.10.18/Jpeg/CamImg1.jpg - Windows Internet
http://192.168.10.18/Jpeg/CamImg1.jpg
File Edit View Favorites Tools Help
http://192.168.10.18/Jpeg/CamImg1.jpg

By then sending this jpg to an out-of-process MATLAB server object, I was able to leverage MATLAB's excellent ability to manipulate images. Using techniques I learned in CSE 573, through Dr. Peter Scott, to perform several image processing activities, I was able to:

- edge detect for the purposes of obstacle avoidance
- detect objects by color (and in the future, size)
- follow moving objects
- home in on a moving laser pointer

Here is an example of edge detection in action. It uses a canny edge filter on the image and returns the edge detected image back to the program for processing. In the following figure, the raw image is on the left captured directly from Rovio, and the processed edge-detected image is on the right:



It is important to note, that although these image processing techniques, and how they would be used by a program, are sophisticated, they are not considered cognitive as defined by our seminar. Instead they could only be considered part of a cognitive robot if they are accompanied by an integrated reasoning system. After being challenged to have the robot chase a red ball, and successfully completing that task, I was educated that such a system is still not cognitive.

I was discouraged to find from our seminar that even a vision system that would integrate color, size, and shape detection and maintain a persistent map of the world surrounding the robot would not be considered cognitive. I was encouraged to suspend my efforts on the vision system to focus more on the reasoning aspect of cognitive robotics.

3.2. Speech Recognition

Another advanced task I was eager to have Rovio perform was speech recognition. This two step process captures streaming audio from the microphone on the Rovio and then manipulating it into a form that the speech recognition engine could use. Though any speech recognition engine could be utilized, I chose Microsoft Speech Recognition, since it has a good success rate and the grammars can be easily manipulated through .NET.

By first opening up a streaming audio connection using the “GetData” cgi script, and then performing a series of RSTP protocol steps [MSDN 2009, see Protocol Details] for setting up the actual RTP audio data, I was able to capture a real time stream from the Rovio microphone. Unfortunately, the Microsoft Speech Recognition engine method to utilize that stream seems to be broken, but I was able to work around this problem by capturing 2 second segments of audio, and processing the captured audio data in these 2 second chunks. The only drawback to this “workaround” method is that if the verbal command spans the segment boundary, recognition of a sentence in the grammar will be nearly impossible. (To offset this possibility, I found that by turning on the headlight when recording occurred helped time the verbal command from the user.)

As can be seen through the videos posted on “RovioWrap” on YouTube.com [Go to http://www.youtube.com/watch?v=PbVQpH4pQeA or you can click to them off the RovioWrap open-source website at http://www.codeplex.com/roviowrap/], Rovio performs nicely with about a 90% recognition rate for a varied group of verbal commands including: move left/right, slide left/right, speak, hello, turn/spin around, goodbye, sit up/down, et al. I was surprised that most of the time a command was spoken, regardless of the orientation of the microphone, that the command was recognized correctly.

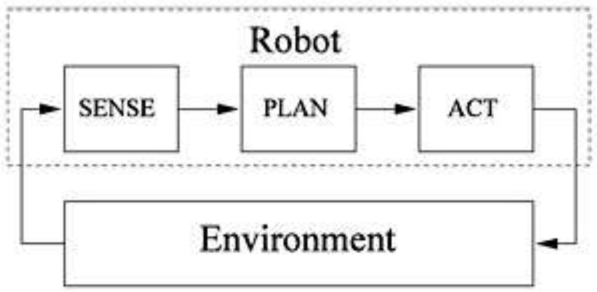
The “RovioDog”, as I nicknamed it, used the Rovio’s audio sensors and motor actuators to interact with the user. Though an integrated reasoning environment was not attached to the program, the states of the program can be *reflected* upon by internal rules. It was able to manage its own mental states through a finite state machine, transitioning between mental states when either specific audio or visual input was provided or when internal rules caused a transition based on internal states. The RovioDog program was able to choose what and when to perceive, fire rules based on those perceptions, and then either decide what behavior to activate, decide whether or not to interrupt a current behavior, or decide if it needed transition to different mental state.

4. Implementation Framework

One of the requirements for a cognitive robot is to have an implementation that is able to embed the sense-reason-act (SRA) cycle in a "uniform theoretical and implementation framework" (principle III). Though I was mildly familiar with the SNePS GLAIR architecture, I was more familiar with the standard ways of implementing a sense-plan-act (SPA) cycle and had my own ideas to explore. Through seminar discussion and examining previous SNeRG documents, I soon discovered that the GLAIR architecture was exactly what I was looking for. I include here, however, an investigation of the other models that became important to my understanding.

4.1. The Sense, Plan, and Act Cycle

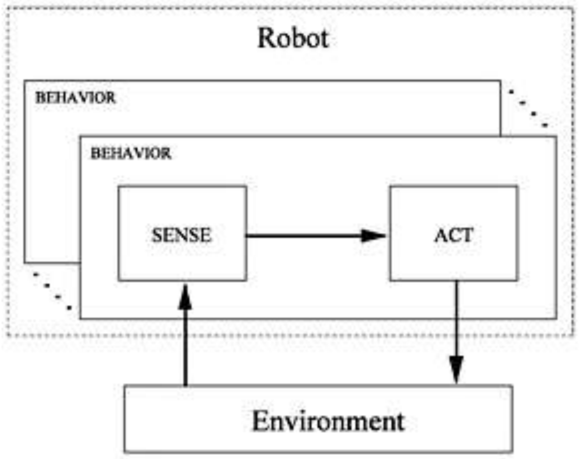
The Sense, Plan and Act Cycle (SPA) is a well known model for implementing robotic behavior. As shown in the figure below, sensing occurs through the sensors (Infra-red motion sensors, sonic distance sensors, camera, microphone, movement, haptic, etc), the sensor values are passed to the planning/controlling module which considers both the sensor input and an internal state (a program or programs that process rules), then various effectors or actuators (motors, sirens, servos, LCD screens, etc.) are activated based on the output from the control module. This SPA cycle continues indefinitely. Though there is no restriction in the model for doing things in parallel, most implementations perform these steps in sequence in a deliberative manner.



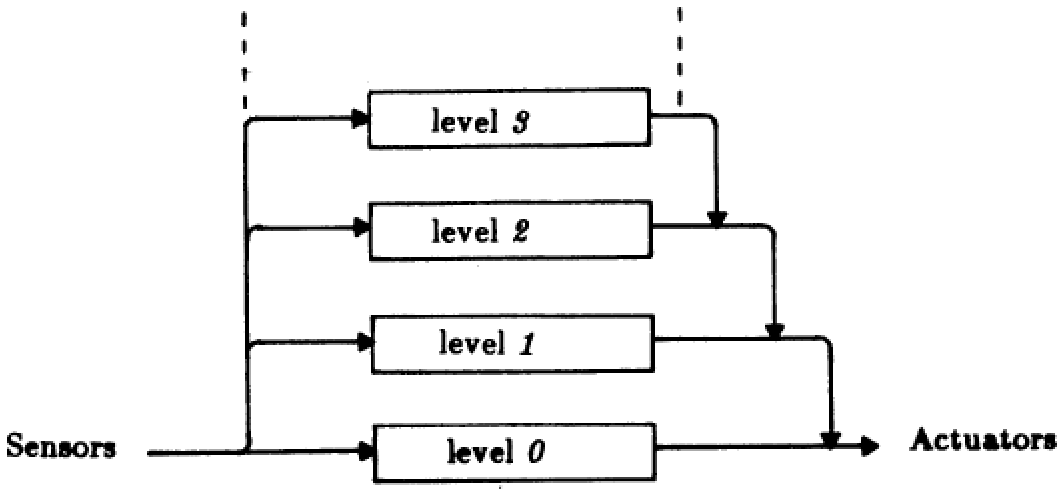
This classic horizontal architecture is sometimes represented as a perceive-reason-act cycle as well. It has several drawbacks, but most notably is the idea that "acts" are perfectly executed. That is to say that when a primitive act is activated, there is no feedback to the primitive act, no further decision making based on sensory input. [Mackworth 2009, p 10-11]

4.2. Subsumption Architecture

A departure from this SPA loop was popularized by Rodney Brooks from M.I.T. in what is called the subsumption architecture [Brooks 1985]. In this architecture, as shown in the figure below, sensory information was tightly coupled with the actuators thereby providing a behavior not seemingly governed by any database of rules. When one behavior becomes dominant due to a stimulus, it interrupts the output of a currently running behavior, and substitutes its own output. Essentially, one behavior is able to subsume the other when necessary, creating a "behavior fusion" [Hexmoor 2008, pg. 3], and this can cause a seemingly intelligent overall behavior.

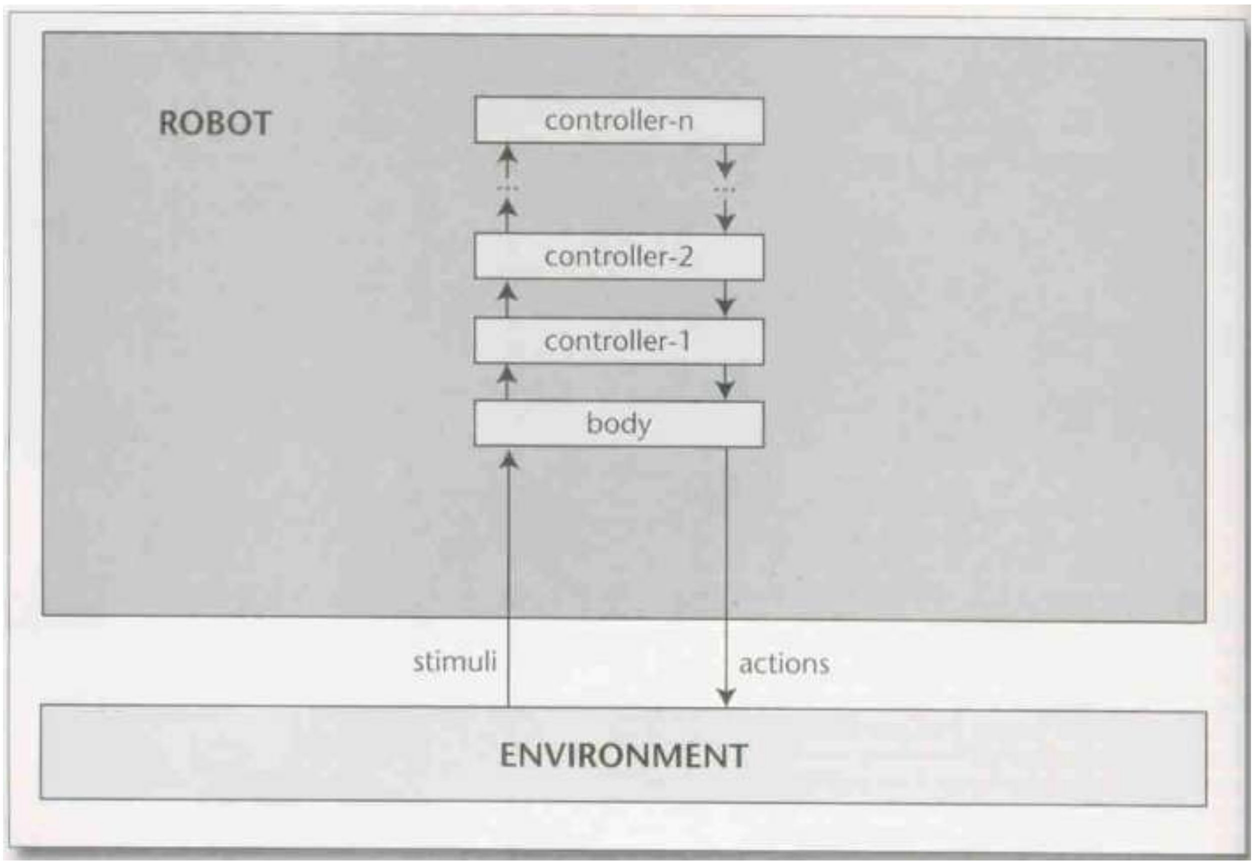


Sense-Plan-Act loop diagram. [Lin 2007]



Subsumption Architecture. [Brooks 1985]

I would suggest that although there is no controlling software, there are controlling circuits. The values that are "sensed" are brought into the circuit and in a way "symbolized" by voltage values or register values, "reasoned" through the use of switches and gates, and "acted on" by the triggering of motor controllers to move the wheels or legs. The overall result is a seemingly synchronous operation of motors that appear to be being controlled directly by environmental conditions, but in fact is just an expression of a logical program in solid state components and circuitry. The environment is still being represented internally, even if the current state of the machine is not considered, there is a logic to what is occurring.

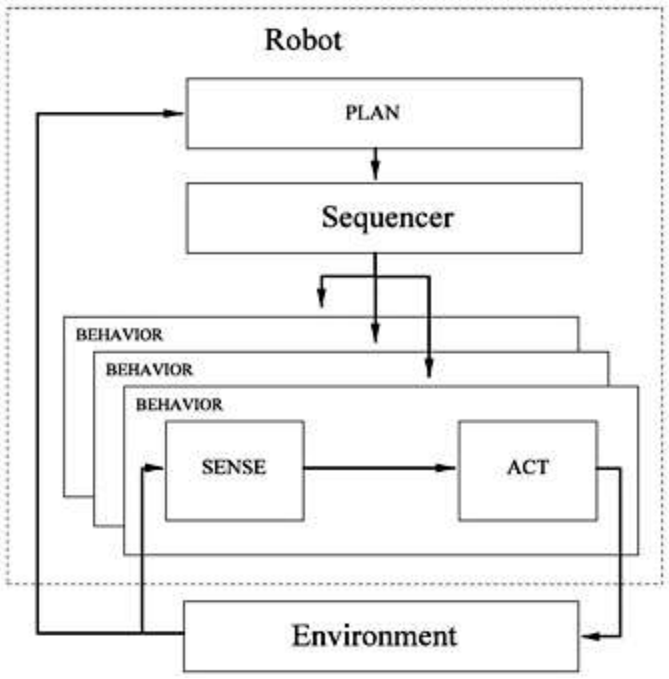


Layering of controllers. [Mackworth 2009]

This vertical architecture can also be thought of as the layering of controllers, each controller directing the "virtual body" below it. [Mackworth 2009, p 21] As one climbs higher in the model, each controller has differing goals based on differing perceptions and models and tend to work in a timespace that lengthens the higher we look. Though this model may mimic the way biological systems have evolved, I would suggest that coordination between the layers could get complicated.

4.3. Hybird Architecture

Another architecture that is being use is a hybrid system that consists of three layers and has properties of both the planning of a SPA and the reflexive subsumption architecture. First there is the *reactive layer*, which tightly couples the sensors to the actuators and provides the reflexive behaviors required of a real-time embodied agent. Next, the *deliberative layer* is what we think of as doing the planning and reasoning. It provides the ability to manipulate a model of the world, as well as carry out high level planning, reasoning about other agents, and other high level "conscious" functioning of the robot. Lastly, the *sequence layer* communicates between these other two layers, translating and expanding sub-goals from the deliberative layer into a sequence of reflexive behaviors, managing the proper sequencing of these behaviors, and reporting success or failure of these sub-goals back to the deliberative layer.

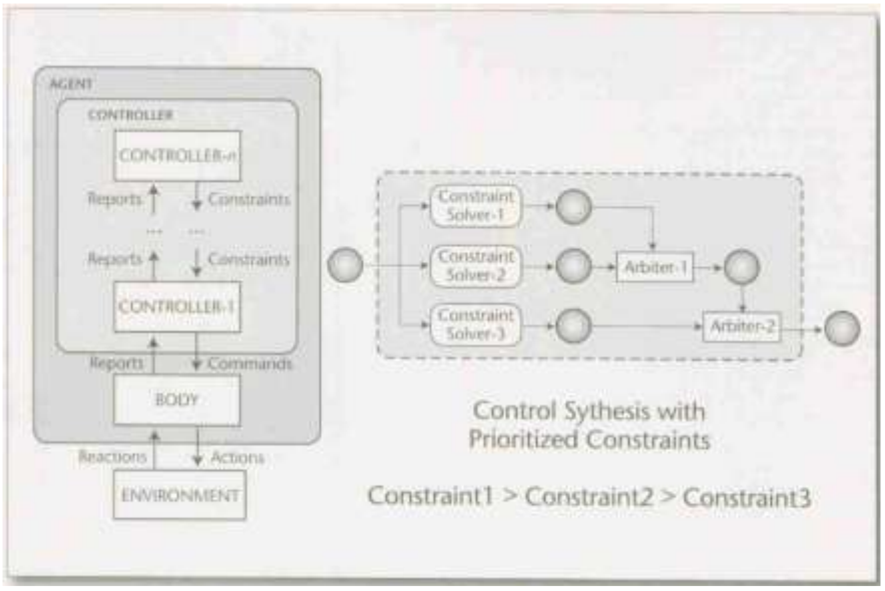


Hybrid Architecture. [Lin 2007]

As can be seen from the figure above [Lin 2007], the SPA steps are still being used, but an additional sequencer step removes some of the sub-goal management from the planning step as well as allowing the robot the ability to act reflexively without "conscious" decision making.

4.4. Constraint-Based Architecture

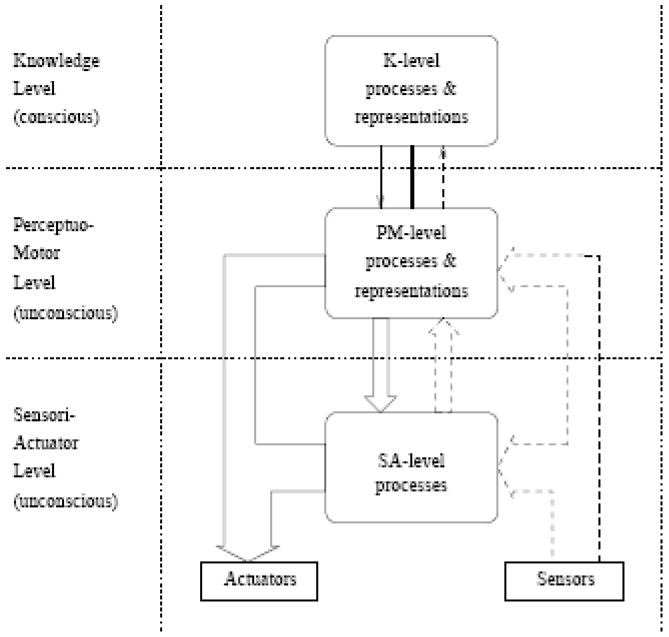
Dr. Alan Mackworth describes another popular model of robotic control that focuses on constraint satisfaction. This is different from the "good old-fashioned AI and robotics", which he calls GOFAIR, which uses the horizontal sense-plan-act model described in section 4.1 often addressing one problem at a time. Instead, the constraint satisfaction problem (CSP) paradigm, focuses on satisfying multiple requirements at a time, merging behaviors like a subsumption architecture, but also is being capable of reasoning at a higher levels and reacting to stimulus at a lower levels. A simple model that Mackworth describes as "a solution" not "the solution" is shown in the figure below.



Constraint Satisfaction Robotic Architecture. [Mackworth 2009]

4.5. GLAIR Architecture

Although my search for comprehensive dynamic robot architecture led me to investigate several paradigms, I found what I was looking for from our own SNePS archives: the GLAIR architecture [Hexmoor, Lammens & Shapiro 1993]. As shown in the figure below, it consists of three layers: the Knowledge Level (KL), which performs reasoning about the world and generates acts or act plans; the Perceptuo-Motor Level (PML), which provides an interface between the KL and the world and/or between the KL and the SAL, but more importantly represents in relation to the body in the world, all the sensory and motor perceptions; and the SAL, or the Sensori-Actuator Level, where primitive motor or sensory acts at the level of the actual hardware are performed. The interaction between these layers is very similar to the hybrid system described in Section 4.3., however, the GLAIR model also allows for the ability to "perceive" environmental sensory data differently than it actually is. This "preprocessing" of the data in an intelligent, but sub-conscious way, closely matches the way humans process information. This would mimic the ability to hallucinate or "fill in the blanks" in a visual or auditory image the way humans and other animals do. I personally find this important; though I'm sure others do not.


Knowledge Level (conscious)
Perceptuo-Motor Level (unconscious)
Sensori-Actuator Level (unconscious)
K-level processes & representations
PM-level processes & representations
SA-level processes
Actuators
Sensors

GLAIR Architecture. [Hexmoor, Lammens, and Shapiro 1993]

Another benefit to the GLAIR architecture is the ability to feed sensory information back directly to the SAL and PML. Closing the loop like this allows the primitive acts called in the PML to react more to its environment, as well as providing a way to implement reflexive actions yet report them up through the layers to the KL.

One could potentially make the argument that these three different layers represent nothing more than three SPA loops, where both the environment and the direct layer below represents the world. Doing so, however, would obscure the duties that these individual layers are required to perform.

Note that the Knowledge Layer, which represents reasoning, planning, and in a larger sense, consciousness, is not directly connected to the environment. All representations that the KL will use will come from the PML; however, not all of what is represented in the PML will be used in the KL. The PML is responsible for what we would deem to be subconscious, everything from perceiving the world, like size, colors, to also coupling its behaviors to its stimuli, to performing sub-behaviors that are required to complete more comprehensive behavior requests.

5. Development of a Cognitive Rovio

I chose to implement my project differently than the other students, but with the full intent to satisfy the three principles of a cognitive robot.

5.1. Satisfying Principle I : An Integrated Architecture

It can probably be argued that an architecture that integrates perception, reasoning, and acting can be implemented in many of the architectures outlined in section 4. My criticism is that not all these varied paradigms allow for modularity nor would they scale well. I feel modularity is important unless we wish to deal with a tangle of rules and varying levels of sensory and perceptual data all at the same level. By keeping the representations to a minimum and confining them to the level at which they would be useful, we reduce the complexity of the implementation.

I greatly admire the GLAIR architecture and felt that it provided the most flexibility and modularity to the implementation. In my survey of the various architectures used in general robotics, I found that I could better satisfy the other two principles of a cognitive robot with the GLAIR model. Therefore, I implemented a GLAIR.NET object, as a thread-safe, multi-threaded .NET object.

By choosing to implement the robot using GLAIR, I understood that I would be required to constrain various tasks, levels of processing, and representations to their respective layers of the model. It would in essence require me to:

- Separate behavior implementation (PML) from reasoning (KL)
- Constrain sensory interpretation to the PML layer
- Implement reflexes in the SAL layer

5.2 Satisfying Principle II : Include Higher Level Cognitive Functions

A higher level cognitive function appears to be a function that manipulates a knowledge representation structure at a level that is "far" removed from low level sensory data. It appears that it can have the same result as a low-level If..Then statement, a simple lookup table, or even a simple program. I would imagine that things like driving, conversation, playing games, and being creative could all be considered a high-level cognitive function.

I also think we can agree that the "wander" routine, being able to navigate through an uncertain landscape without colliding with anything, could be considered a high-level cognitive function.

Though lower level creatures such as bugs and fish can perform this task, the fact that we as humans can do it with deliberation and reflection should make this a high-level task.

One of the discussions we had in our seminar was a comparison of two different "wander" routines, one written in pseudo-code any-language and one written in pseudo-code SNePS. The question was, if they both perform the specific behavior, then what makes one cognitive and the other not?

Pseudo-code for Wander routine	**SNePS Pseudo-code for Wander routine**
Loop {	
If ObstacleInFront Then	WheneverDo((ObstacleInFront)
TurnRight 90 degrees	(TurnRight(90)))
Else	
MoveForward	WheneverDo((Not ObstacleInFront)
	(MoveFoward))
}	

It became clear to me from the discussion, that each code base that it was in essence a loop with a few conditions and actions. When I asked the question as to why the SNePS code is cognitive, the reply was that it is situated within a framework that can answer questions about the code and the state of the running program as well as the fact that the SNePS system as a whole is able to create new routines from reasoning about the older routines. So I think I would have to conclude that the behavioral code itself is not what makes something cognitive, but it is the system that it appears in.

In essence, the system of “me” and the “any-language pseudo-code” could be considered equivalent to the system of “SNePS” and the “SNePS pseudo-code”. This is important to implementation in that it just means that a cognitive robot is neither its code nor its coded behaviors, but instead it is what the entire system can do WITH said code within in its reasoning process. As long as the system can explain why the code works in relation to itself and the world, then it is cognitive.

I had chosen to implement this interpretation and reporting as a visual interface. It exposes the inner states of the program and reports the reasons for the changes of mental states, as well as providing an episodic memory archive to see what perceptions causes which mental changes. It acts like the “me” or the “SNePS” system described in the above example, or the liaison between the user and the code, with the ability to report the state of the robot and about what it is reasoning about. Through some seminar discussion, it was decided that this would not be acceptable and I should not pursue this.

5.3. Satisfying Principle III : Reasoning About Things

Reasoning, for this seminar, has a particular implementation in mind for cognitive robotics; it is concerned with logical inferences. Antecedents and consequents fitted into knowledge representation structures are applied to other knowledge representation structures through a series of pre-defined introduction rules, and gradually alter the state of the knowledge base. As the consequents are triggered, they can activate various primitive functions tied to the hardware of the robot, thus enacting what we would identify as a behavior from the robot.

To call a robot implemented in this way a *cognitive* robot, however, would be premature. We must also satisfy principle III and reason about specific ideas and concepts, for without them, the robot would only be a series of conditional statements and loops, modifying values in variables or memory structures. These "things" that must be reasoned about appear important to cognition because they appear to be things that are specific to human reasoning, but more importantly, they are essential for both complex tasks and verbal communication.

5.3.1. Goals

A goal, in simplistic terms, is a desired state of the world. A cognitive robot would therefore need to determine how it needs the world to look and how it would be situated in that world, before (or during) the actions to change the world into that goal state have begun.

Initially, I intended to implement this in terms of a goal stack. The idea would be that when the robot determines what the world state needs to look like (through user request or the changing state of the world), it will place that constraint on the goal stack. The robot is then able to reason about that higher level goal, to determine any sub-goals that are needed, and so on. Each goal on the stack will be reasoned about until such time a sub-goal has a sequence of primitive acts/behaviors associated with it, at which time the PML will be called with the appropriate request.

It was noted during the seminar that undertaking such an implementation was outside the scope of the course as well as being too difficult to implement in the time remaining. I suspended my development on this for the future.

5.3.2. Actions

A primitive action is performed in the PML layer. The intended effects of that action may be different from the actual effects of that action. I have divided the primitive action into two types in order to deal with the model I have defined for goals and cognitive state. I have also placed a few restrictions on how the PML layer should deal with a request to perform an action.

I have defined two types of primitive actions, one is an (intended) physical manipulation and the other is an instantaneous mental manipulation. The difference between the two types is based on how long the manipulation will take and whether or not feedback is required. If duration or feedback is required, additional sub-actions can be called (this will be explained below).

For the instantaneous mental manipulation, there is an instant effect on the cognitive state. This can include the changing of a state variable, but does not allow for feedback from the SAL, PML, or KL.

For the physical manipulation, there is a potentially gradual change to the environment or cognitive state. Assuming that the cognitive system can work faster than the action, there is potential for a primitive action to be called again and again while it is in progress. In addition, there may be need for that primitive action to be stopped if another primitive action is required more urgently. To address these concerns, I have added two additional features to a primitive action: an “in-progress” flag and a “stop” interrupt.

Please note that the physical manipulation type of primitive action does not require a physical sense or act to occur. Instead, it includes any gradual change to the cognitive state, which means, a primitive action that alters a cognitive state over the course of an entire day can be easily implemented here.

By design, the physical manipulation and mental manipulation primitive actions cannot be called by the same state-change rule. Only one primitive action can be called by each rule, thus episodic memory would be made up of sub-vector, action pairs.

5.3.3. Perception

Although sensing is something that is performed automatically at the SAL layer, perception is performed at the PML layer. When the KL layer (or the PML layer in the case of feedback within a primitive action) directs its attention at a specific percept, this would be a deliberate act of perception. The KL layer decides through a rule where to direct its attention, whereas the

PML layer within a primitive action in a feedback situation can direct its attention directly to the raw sensory data or the perceived data.

For example, speech recognition is being done every two seconds in this implementation. The PML layer automatically processes the audio data from the SAL layer and “perceives” it as text. When the PML layer completes its processing of the audio and has a perception to report, it sets a cognitive state variable called “SpeechPerceived” to VERY HIGH (200). Several rules in the KL use SpeechPerceived in the antecedent, and all may be able to conclude their consequent. All rules use Boolean short-circuiting.

Let us look at some of these rules:

Rule 1. This rule decreases the perception of the speech over time. It simply decreases the SpeechPerceived state from VERY HIGH (200) to VERY LOW (50) over an arbitrary four seconds.

Rule 2. This rule determines if the current goal includes an active physical act and if so, directs its attention to see if speech can be perceived and if so determines if someone is saying “STOP”. If this is all true, then the cognitive state variable “StopBehavior” is set to VERY HIGH.

5.3.4. Cognitive States and Emotions

I have probably taken liberty with the term "cognitive state" in the previous sections. I describe it more as set of variables, which make up a vector. The vector is considered to be the cognitive state of the system. This has the potential to cover all states of a system.

Cognitive states also include emotions. I have up till now considered an emotion just to be another cognitive state and so it is indeed reasoned about using both state-change rules and episodic memory rules. I *could* define the "SpeechPerceived" state from the section above at an emotional level as the "Need to find out what was said" emotion, or perhaps I could have another more global state that matches what us humans would consider "curiosity". Although I may be incorrect, I feel that a cognitive state is interpreted by the system it operates in.

Reasoning about cognitive states of other agents would require a modeling of the other agents. I have implemented the robot to respond to its perceptions of the world around it, and currently the perception of other agents (which would need to be developed in the PML) has not been included. The level at which I stopped development of PML layer sensory-perception routines was at color blob identification and location in a visual field. Though I have speech recognition implemented, it does not represent that the speech is anything other than a noise and is not "caused" by anything in particular.

A next step in the development of this system would be the development of representations of objects within a world representation. Once objects are represented, then an object as an agent can be realized. Finally, once an agent is represented, we can attribute cognitive states and emotions to the agent and then reason about it.

5.3.5. Time

The robot keeps a 24 hour time stamped record of its mental state, primitive actions taken, raw sensory data (that has been part of a perception act), and some perceptions (perception results). Currently it has an episodic memory of a single day, which on average will generate an estimated 2.6 gigabytes of data.

The concept of time within the framework of reasoning is currently constrained to expectation states in the system. Activity levels are broken into 5 minute blocks. The PML examines the next 5 minute block and the next size 5 minute blocks and determines the average activity level from 24 hours ago. Should the examined durations contain a lot of state changes, the “ExpectedActivity5Minute” state and the “ExpectedActivity30Minute” state is set to HIGH (150). These expectation states dictate if the robot will go into a wander mode or come off the charging station.

6. Project Problem and Implementation

We have been tasked to come up with an implementation of a cognitive robot that will solve a high-level cognitive function. Whereas most students in the seminar are attempting a complex behavior, much of which will not necessarily address all three principles of cognitive robotics, I am attempting to address the principles exactly for the purposes of promoting understanding for myself.

There are two sections, one in which I will describe how to use the PML (the RovioWrap object) and the other in which I will illustrate a simple KL application, simulating the behavior of a robotic dog, called "RovioDog". An examination of the rest of the seminar's student projects will quickly reveal that the RovioDog implements more of the KL than any other (no other application represented cognitive states or rules that transition between them). It can be argued successfully that RovioDog is *not* a fully cognitive robot by the definition given at the beginning of the paper.

The full code and working implementation for these sections can be found in perpetuity at: http://www.codeplex.com/RovioWrap , complete with detailed in-line commenting and links to various video resources demonstrating the working applications. (I am "Sidekick" who owns the open-source project, feel free to email me if you would like to be one of the administrators who extend the code).

6.1. Using RovioWrap

I will not be giving a step by step tutorial on how to program in .NET or integrate the RovioWrap object in your program. RovioWrap is a collection of primitive actions that directly communicate to the SAL of the robot. The SAL code is hidden behind the web server of Rovio, however, through HTTP requests (as described in Section 2), I developed functions that manage the communication between the KL and the SAL. (For more information about SAL, PML, and KL, please see the description of the GLAIR architecture described in Section 4.5]

As it is a .NET object, it can be instantiated external to VB.NET and in any other .NET language or even non .NET language running a WOW32 (Windows 32bit layer). This includes MONO and WINE, both layers that run in Linux and give native Linux programs the ability to run Windows programs. I have forum reports that the RovioWrap.NET object has run successfully, without modification, on WINE. [See http://www.winehq.org/]

I will be giving all examples in VB.NET, though the object has been used in Java, VC.NET, and ACL (Allegro Common LISP). These, and more, examples can be found in the "RovioController.vb" file.

6.1.1. Instantiating RovioWrap

To create an object that can throw events, declare as the following:

```
  Private WithEvents RO As New RovioWrap
```

To connect the RovioWrap object to your Rovio, you can use the default IP and port shown below, or you can change it as necessary. Depending on how you will connect, either using a home network or an ad hoc network with your computer, there are additional steps to get this to work. See your Rovio documentation for further info. My recommendation is an Ad Hoc network, which has been verified as working on Windows, Mac, and Linux OS's.

```
    RO.RovioIP = "192.168.10.18"
    RO.RovioPort = "80"
```

You may need to configure a username/password before using the object in any way. The default username/password is empty.

```
    RO.Username = "Scott"
    RO.Password = "LetMeIn"
```

6.1.2. Primitive action

One of the many primitive actions implemented is the "RaiseHead" act. You can call the PML with the RovioWrap object in the following manner:

```
      RO.HeadElevationState = 2
```

This will raise the head up, but you can also lower the head or move it half way up by changing the 2 to a 0 or a 1, respectively.

Other movement actions include moving the Rovio forward, backward, side to side, or even spin it around. For example, to move the Rovio forward at speed 1:

```
      RO.Move(RovioWrap.MovementParameter.moveForward, 1)
```

To move it left or right, at a speed ‘s’, you can call either of the following:

```
RO.Move(RovioWrap.MovementParameter.rotateLeftBySpeed, s)
```

or

```
RO.Move(RovioWrap.MovementParameter.rotateRightBySpeed, s)
```

Not all primitive physical actions are movements. You can also have the Rovio turn on or off its headlight or even play a .wav file. To turn on the headlight, call:

```
RO.Headlight = True
```

Call this code to play a .wav file (make sure the .wav file is in the proper format for Rovio – PCM encoding, 8 mhz, 16-bit mono):

```
RO.PlayAudioFile("c:\rovio\Scott_HelloThere.wav")
```

6.1.2. Primitive sense action

Another type of primitive action is a sense action. You can get the current status of the IR (Infra-red) obstacle detection sensor by calling the following primitive action:

```
RO.GetStatus()
If RO.ObstacleFlag Then
```

The “GetStatus” call is needed to fetch the current sensor readings. Readings are not taken automatically, instead the PML or KL *decides* when to attend to a perception. Though the

GetStatus call is a sense act, it only gets the raw sensory data, so you will also need to call "ObstacleFlag" in order to *perceive* what is going on. [1]

To grab the current image that Rovio can see, you can call the following method:

```
    PictureBox1.Image = RO.CurrentImage
```

This code places a .jpg in the Windows picture box object. You can then manipulate this image directly, or save the image to a file like so:

```
    PictureBox1.Image.Save("c:\rovio\ImageIn.jpg")
```

6.1.3. Advanced sensing with vision

One of the more interesting aspects of vision is processing the image to extract information. A tool I used to do this uses a vast array of image processing functions packaged in a separate application called "Matlab". With .NET, it is easy to interact with Matlab in a quick and efficient way, just create an object of a Matlab application, and call it as if you were typing on its command line. For example:

```
  ' Create the Matlab object
  Dim Matlab As Object
  If Matlab Is Nothing Then
```

[1] Here is a quick implementation note concerning obstacle detection on the Rovio. You will have problems getting accurate readings based on the flooring you are running on. More "specular" flooring, like nylon carpet or cement, may reflect more IR rays than you would assume. It would be advisable to use sensor fusion and use this value as one of several to decide if there is an obstacle. An approach I took was to use edge detection, however this would not work well with walls.

```
    Matlab = CreateObject("Matlab.Application")
  End If

  ' Change the directory to where the script is
  Matlab.Execute("cd c:\rovio\;")

  ' Execute the "DetectEdges" script
  Matlab.Execute("DetectEdges;")
```

Matlab will then use the “ImageIn.jpg” created when getting an image from Rovio (See section 6.1.2) and output an image to another file “ImageOut.jpg”. You can, of course, change the script or write your own scripts. Some additional scripts that can be found with the source code are “LocateLaser”, “LocateRedBall”, and “LocateBlueBall”.

6.1.4. Advanced sensing with speech recognition

Another advanced sensing technique is implemented in RovioWrap. I encourage you to look through the code to get familiar with the procedure, but in a nutshell what happens is the following:

1. Establish an RTSP connection with Rovio
2. Acquire an RTP stream from Rovio
3. Capture the RTP packets representing the real time audio from its microphone
4. Place that audio data in a stream
5. Send that stream directly to Microsoft Speech Recognition (MSR) engine

In RovioDog, the primitive sensory act occurs over a 2 second period at which time the audio data stream is sent to a file, instead of directly to the MSR engine. Then the MSR engine loads the file and reports if it has found a recognized phrase. The procedure and code is too lengthy to explain here, but hopefully this will get you on the right track.

However, in the interests of getting you up and running more quickly, you can just call the following method to record '2' seconds of audio:

```
RO.RecordWAVFile(2, "c:\rovio\CapturedAudio.wav")
```

And then, after setting up the MSR engine with a grammar (See TrainedRovio.vb for more information), you can merely call:

```
SR.SetInputToWaveFile("c:\rovio\CapturedAudio.wav")

Dim Result As System.Speech.Recognition.RecognitionResult
Result = SR.Recognize()
RecognizedPhrase = Result.Text
```

At this point, RecognizedPhrase would include the text phrase that was recognized from the speech. Be sure to create a grammar (as well as the speech recognition, "SR" object) before calling the MSR engine, but essentially it is this simple.[2]

[2] Beware sending a stream directly to the speech recognition engine. The MSR engine has some trouble with custom streams. I have documented this problem in the code as well as on the forums, but to date there is no solution, which is why the "write to file" then "recognize from file" is the only workaround that worked at this time.

6.2. Running RovioDog

RovioDog is implemented in the file, “TrainedRovio.vb”. You can select this to be run from the RovioController screen menu options, or change the project to start up with this form. Upon inspection of the code, you can see that it is a simple finite state machine implemented in a Select called every 100 milliseconds from a timer. (See the Timer1_Tick event.)

In this timer, we can see that the mental state of the RovioDog is maintained by one variable called “State”, along with several sub-states to manage the subtasks associated with each state.

At first, State=1 and represents the “Turn on” state. This is the state that the RovioDog is started with. From here you can note that the RovioDog will immediately speak a welcome phrase and then wait for a command in State=2.

State=2 represents the state that the RovioDog is ready to accept a verbal command. When other primitive actions are being executed, RovioDog cannot be in State=2 in this program, however, there is no restriction in the RovioWrap that limits speech recognition when no primitive actions are taking place.

State=21 is the state where the perception of speech is processed. If there was a recognized phrase, we will see that there is the transition to an “understanding” state (State=5), but if there was no recognized phrase, then the state quickly transitions back to State=2 in a few timer ticks.

The “understanding” state, is the state that occurs if speech has been perceived. State=5 will examine the speech and decide whether to take a specified primitive action, or begin a behavior that requires a further sense-reason-act cycle. For example, if the phrase “Wander” is perceived, then the next mental state will be State=30, which includes a behavior called MoveRovio.

MoveRovio consists of calling several primitive actions, a sense action and the appropriate movment actions. Whether MoveRovio should be considered part of the PML or the KL, I think is debatable. Since the MoveRovio routine calls other primitive acts, it could be either, but since it makes a decision whether to turn, move forward, or speak depending on the result from the primitive sense act RO.ObstacleFlag, that could be considered reasoning.

Additional functionality is included and this model can be extended indefinitely. Though the model admittedly does not have the power of a full reasoning system, like SNePS, will a little modification it can report what it is doing and why it is doing it. In fact, if one examines the logs, a complete history of the what and why can be discovered. I do not think that this satisfies fully the principles defined at the beginning of the paper, but it moves us one step closer to them.

For further information, and video of a demonstration, please visit “RovioWrap” on YouTube.com [Specifically http://www.youtube.com/watch?v=PbVQpH4pQeA] or you can click to them off the RovioWrap open-source website at http://www.codeplex.com/roviowrap/.

7. Conclusion

I have learned a great deal during this seminar, both about cognitive robotics and cognition in general. I apologize for bringing in ideas that may have been too far off topic in the interests of sparking discussion and generating brainstorming sessions. I further apologize if my enthusiasm for the topic came across as arrogant, as that was the furthest from my intent. I am afraid my comfortableness with both the topic and the participants whom I have known for years in the seminar made me feel safe to speak my mind without retribution and thus an over-zealous tongue may have caused the people that I have the utmost respect for to misinterpret my actual state of mind.

With these disclaimers aside, I feel that cognitive robotics is a difficult concept to address in three months. If cognition is the main goal, robotics is a distraction. There are so many other concerns when dealing with robotics, that it is easy to lose the focus of creating a "mind". The addition of the various physical application issues when dealing with sensing and acting makes the cognition task inaccessible until these are adequately addressed.

I personally found it hard to conceive of a non-trivial cognitive task doable within the framework of the primitive perceptions we had available. I would highly recommend that at the very minimum, there would be primitive visual color and shape detection routines already in place in the PML. Better yet, an integrated, even if simple, viable object detection system would allow the students to expand the scope of their project and have a *specific need* for a KL. In other words, dealing with tangible objects would *require* a layer of reasoning that a simple wander program does not.

Concerning specifically the Rovio platform, though it is very inexpensive for the amount of work it does, it lacks a few sensors that I think are pivotal. The one temperamental IR sensor makes it very difficult to do anything with certainty and prevents an application programmer from thinking bigger. I would recommend modifying the Rovio to include bump sensors (or perhaps capture audio from the microphone to detect a “bump”) or change the IR sensor to be less sensitive (perhaps by decompiling and modifying the Rovio through a patch).

The RovioWrap open-source project [http://www.codeplex.com/RovioWrap] that I have created has been live for several months now and is quite popular with other amateur roboticists. To date, there have been over 1000 unique visits to the web page and over 340 downloads of my code. I am appreciative to all the forum members and personal emails that I have received encouraging me to continue my efforts.

I would like to thank Dr. Shapiro for holding the seminar and I would like to thank the other students for their enthusiasm and discussions on the various topics that came up. I was thankful for whatever discussion I was able to have with you all and hope to continue such discussions in the future.

References

1. Brooks, R. A. 1985 A Robust Layered Control System for a Mobile Robot. Technical Report. UMI Order Number: AIM-864., Massachusetts Institute of Technology. *http://people.csail.mit.edu/brooks/papers/AIM-864.pdf*

2. H. Hexmoor , J. Lammens, S. Shapiro, 1993. *Of Elephants and Men*, In The Biology and Technology of Intelligent Autonomous Agents, Luc Steels (Ed.), 312-344, Springer. [PDF]

3. Hexmoor, Henry, 2008. "High level software control loop". Retrieved April 2009 from http://www.cs.siu.edu/~hexmoor/classes/CS404-S09/Hexmoor-control-loop.pdf

4. Lin (ed), Hong. "Chapter XVII - Robust Intelligent Control of Mobile Robots". Architectural Design of Multi-Agent Systems: Technologies and Techniques. IGI Publishing. © 2007. Books24x7. *http://common.books24x7.com/book/id_20788/book.asp*

5. Mackworth, A. (2009). Agents, Bodies, Constraints, Dynamics, and Evolution. AI Magazine, 30(1). *http://www.aaai.org/Library/President/Mackworth.pdf*

6. Microsoft Developer Network (MSDN), 2009. Real-Time Streaming Protocol (RTSP) Windows Media Extensions. Retrieved January 2009, from http://msdn.microsoft.com/en-us/library/cc245238(PROT.10).aspx

7. Stuart C. Shapiro, FevahrCassie: A Description and Notes for Building FevahrCassie-Like Agents, SNeRG Technical Note 35, Department of Computer Science and Engineering, University at Buffalo, The State Universtiy of New York, Buffalo, NY, September 26, 2003.

8. Stuart C. Shapiro and Michael Kandefer, A SNePS Approach to The Wumpus World Agent or Cassie Meets the Wumpus. In Leora Morgenstern and Maurice Pagnucco, Eds., IJCAI-05 Workshop on Nonmonotonic Reasoning, Action, and Change (NRAC'05): Working Notes, IJCAII, Edinburgh, 2005, 96-103.

9. Tim Jones, M.. "Chapter 10 - Robotics and AI". Artificial Intelligence: A Systems Approach. Infinity Science Press. © 2008. Books24x7. *http://common.books24x7.com/book/id_20698/book.asp*

10. WowWee Group Limited (n.d.) October 8, 2008 API Specification for Rovio Version 1.2. Retrieved January 2009, from http://www.WowWee.com/static/support/rovio/manuals/Rovio_API_Specifications_v1.2.pdf